UNIVERSIDAD CARLOS III DE MADRID

TESIS DOCTORAL

MODELADO AUTOMÁTICO DEL COMPORTAMIENTO DE AGENTES INTELIGENTES

Autor:
José Antonio Iglesias Martínez

Directores:
Araceli Sanchis de Miguel
Agapito Ledezma Espino

DEPARTAMENTO DE INFORMÁTICA

Leganés, Enero 2010

II

Departamento de Informática

Escuela Politécnica Superior
Universidad Carlos III de Madrid

Modelado Automático del Comportamiento de Agentes Inteligentes

AUTOR: José Antonio Iglesias Martínez
DIRECTORES: Araceli Sanchis de Miguel
Agapito Ledezma Espino

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Carlos III de Madrid, el día ..... de ..... de 2010.

Presidente: D. .....

Vocal: D. .....

Vocal: D. .....

Vocal: D. .....

Secretario: D. .....

Realizado el acto de defensa y lectura de la Tesis el día ..... de ..... de 2010 en .....

Calificación: .....

EL PRESIDENTE LOS VOCALES

EL SECRETARIO

A mi madre

“Sólo aquellos que ven lo invisible pueden hacer lo imposible”

Frank Gaines

Agradecimientos

No sólo agradecer sino también dedicar este trabajo a mis padres, quienes me ayudan y apoyan en todo momento, quienes tengo siempre como referentes, y a quienes llevo en el corazón cada día. Mil gracias por todo.

A mis tutores, Araceli y Agapito, responsables de todo lo que he aprendido durante estos años. Gracias por vuestros consejos, ayuda y comprensión. Es todo un lujo trabajar con vosotros.

A todos y cada uno de los integrantes del grupo CAOS; Germán, Paula, Jmaw, Mari Paz, Bea, Raúl, JP, Jorge y Javi. Gracias por vuestra ayuda y por hacerme el trabajo tan agradable. Al resto de compañeros de la universidad, de comedor y de cafés, gracias por hacerme sentir tan bien aquí.

A mis *otros* tutores con quienes he tenido la suerte de trabajar, Giuliano, Kaminka y Plamen. Gracias por hacerme parte de vuestro grupo de investigación y por haberme enseñado tanto.

A mis hermanas, Bea y Pily, a las que quiero con locura, por lo mucho que me han ayudado en todos los momentos de mi vida. A mis sobrinos, Martin y Sira porque me alegran cada momento que estoy con ellos. Y a mi *nueva* sobrina, a la que todavía no ha nacido y ya tengo ganas de abrazar.

A mis cuñados, primos, tíos y amigos de la Gil, de Burgos, de Medina, de Madrid y del Jaleo, a quienes agradezco el que formen parte de mi vida. Sois geniales.

A María, por su apoyo, comprensión y por estar ahí cada día. Gracias por los momentos que hemos pasado juntos y que espero que se repitan por siempre.

A todos, ¡un millón de gracias!

Resumen

Las teorías más recientes sobre el cerebro humano confirman que un alto porcentaje de su capacidad es utilizado para predecir el futuro, incluyendo el comportamiento de otras personas. Para actuar de la forma más adecuada en un contexto social, los humanos tratan de reconocer el comportamiento de las personas que les rodean y así hacer predicciones basadas en estos reconocimientos. Cuando este proceso se lleva a cabo por agentes software, se conoce como modelado de agentes donde un agente puede ser un robot, un agente software o un humano.

El modelado de agentes es un proceso que permite a un agente extraer y representar conocimiento (comportamiento, creencias, metas, acciones, planes, etcétera) de otros agentes en un entorno determinado. Un agente capaz de reconocer el comportamiento de otros, puede realizar diversas tareas como predecir el comportamiento futuro de los agentes observados, coordinarse con ellos, facilitarles la ejecución de sus acciones o detectar sus posibles errores. Si este reconocimiento puede ser realizado de forma automática, su utilidad puede ser muy relevante en muchos dominios. En esta tesis doctoral se aborda la tarea de adquirir automáticamente conocimiento acerca del comportamiento de otros agentes inteligentes.

Actualmente, las técnicas para modelar el comportamiento de otros agentes están comenzando a surgir de forma importante en el campo de la Inteligencia Artificial. Cabe destacar que en la mayoría de las investigaciones actuales, se proponen modelados no generales de un determinado tipo de agentes en un dominio concreto, es decir, modelados *ad hoc*.

Esta tesis doctoral presenta tres enfoques diferentes para el modelado automático del comportamiento de agentes inteligentes basado en la identificación de patrones en un comportamiento observado. Estos enfoques permitirán que un agente situado en un entorno determinado, sea capaz de adquirir conocimiento acerca de otros agentes situados en el mismo entorno. Cada enfoque propuesto posee características particulares que le permiten adecuarse a un tipo de dominio, lo que implica que se puede adquirir conocimiento de otros agentes en diversos Sistema Multiagente. Los tres enfoques propuestos transforman las observaciones del comportamiento de uno o varios agentes en una secuencia de eventos que lo definen. Esta secuencia es analizada con la finalidad de obtener su correspondien-

te modelo de comportamiento. De esta forma, en esta tesis doctoral, la tarea de modelado e identificación del comportamiento de uno o varios agentes es tratado, principalmente, como un problema de minería de secuencias de eventos. La aplicación de cada enfoque propuesto en dominios muy diferentes demuestra su generalidad.

Abstract

There are new theories which claim that a high percentage of the human brain capacity is used for predicting the future, including the behavior of other humans. Planning for future needs, not just current ones, is one of the most formidable human cognitive achievements. To make good decisions in a social context, humans often need to recognize the plan underlying the behavior of others, and make predictions based on this recognition. This process, when carried out by software agents, is known as agent modeling where an agent can be a software agent, a robot or a human being.

Agent modeling is the process of extracting and representing knowledge (behavior, beliefs, goals, actions, plans, etcetera) from other agents. By recognizing the behavior of others, many different tasks can be performed, such as to predict their future behavior, to coordinate with them or to assist them. This behavior recognition can be very useful in many applications if it can be done automatically. This thesis is framed in the field of agent behavior modeling.

Most existing techniques for plan recognition assume the availability of carefully hand-crafted plan libraries, which encode the a-priori known behavioral repertoire of the observed agents; during run-time, plan recognition algorithms match the observed behavior of the agents against the plan-libraries, and matches are reported as hypotheses. Unfortunately, techniques for automatically acquiring plan-libraries from observations, e.g., by learning or data-mining, are only beginning to emerge.

This thesis presents three different approaches for creating automatically the model of an agent behavior based on the analysis of its atomic behaviors. Each approach is suitable for different purposes, but in all of them, the observations of an agent behavior are transformed into a sequence of events which is analyzed in order to get the corresponding behavior model. Therefore, in this thesis, the problem of behavior classification is examined as a problem of learning to characterize the behavior of an agent in terms of sequences of atomic behaviors. In order to demonstrate the generalization of the proposed approaches, their performance has been experimentally evaluated in different domains.

Índice general

1. Introducción **1**
- 1.1. Objetivos . . . 3
- 1.2. Organización de la Memoria . . . 5

2. Estado del Arte **7**
- 2.1. Agentes y Sistemas Multiagente . . . 7
  - 2.1.1. El concepto de Agente . . . 7
  - 2.1.2. Sistemas Multiagente . . . 8
- 2.2. Modelado de Agentes . . . 9
  - 2.2.1. Teoría de Juegos y el Modelado de Oponentes . . . 10
  - 2.2.2. Reconocimiento de planes . . . 13
  - 2.2.3. Modelado de agentes en Sistemas Multiagente . . . 16
  - 2.2.4. Modelado de Usuario . . . 21
- 2.3. Minería de secuencias de datos . . . 24
  - 2.3.1. Definición y ámbito . . . 25
  - 2.3.2. Aprendizaje automático en secuencias de elementos. . . . 26
- 2.4. Sistemas Auto-Adaptativos (*Evolving Systems*) . . . 28
  - 2.4.1. Motivación . . . 28
  - 2.4.2. Definición . . . 28
  - 2.4.3. Diferencia con otros sistemas existentes . . . 29
  - 2.4.4. SAA como Sistemas Borrosos Basados en Reglas . . . 30
  - 2.4.5. Otros enfoques relacionados . . . 32
- 2.5. Conclusiones . . . 35

3. Modelado de Agentes Mediante Comparación de Secuencias de Eventos: M-COMP **37**
- 3.1. Creación de Secuencias de Eventos - MCSE . . . 39
- 3.2. Creación de Modelos de Comportamiento - MCMC . . . 39
- 3.3. Comparación de Comportamientos en modo lotes - MCCL . . . 44
  - 3.3.1. Estructura básica del algoritmo . . . 45

3.4. Comparación de Comportamientos en Tiempo Real - MCTR 46
3.5. Complejidad de la biblioteca BIBMOD 48
3.6. Evaluación: *RoboCup Coach* 51
3.6.1. La RoboCup 51
3.6.2. Competición *RoboCup Coach* 53
3.6.3. Aplicación de M-COMP 57
3.6.4. Configuración Experimental 61
3.6.5. Resultados 63
3.7. Conclusiones 68

4. Modelado de Agentes Utilizando Secuenciación de Eventos: MAUSE **69**
4.1. Creación y almacenamiento de Modelos de Comportamiento . . . 70
4.1.1. Obtención de Secuencias de Eventos - MCSE 70
4.1.2. Creación del Modelo de Comportamiento - MCMC 71
4.1.3. Almacenamiento de Modelos de Comportamiento - BIBMOD 72
4.2. Detección de Modelos de Comportamiento - MDMC 72
4.2.1. Método de comparación de modelos - ABCD 72
4.3. Evaluación: *RoboCup Coach* 74
4.3.1. Aplicación de MAUSE 74
4.3.2. Configuración Experimental 77
4.3.3. Resultados 77
4.4. Evaluación: Interfaz de Línea de Comandos 79
4.4.1. El dominio 80
4.4.2. Conjunto de datos 81
4.4.3. Aplicación de MAUSE 81
4.4.4. Configuración Experimental 83
4.4.5. Resultados 84
4.4.6. Comparación con *Modelos Ocultos de Markov* 85
4.5. Conclusiones 87

5. Modelado Auto-Adaptativo de Agentes con Flujo Continuo de Eventos: M-ADAP **91**
5.1. Gestión de Modelos de Comportamiento Auto-Adaptativos (MGMA) 92
5.1.1. Representación de los modelos en el Espacio de Características 93
5.1.2. Algoritmo ABCD-ADAP 95
5.1.3. Clasificación vs. Agrupamiento 101
5.1.4. Interpretación del clasificador 101
5.2. Evaluación: Interfaz de línea de comandos 102
5.2.1. Conjunto de Datos 102

5.2.2. Aplicación de M-ADAP . . . 103
5.2.3. Configuración Experimental . . . 105
5.2.4. Resultados . . . 108
5.3. Evaluación: Entornos inteligentes . . . 114
5.3.1. Conjunto de Datos . . . 114
5.3.2. Aplicación de M-ADAP . . . 117
5.3.3. Configuración Experimental . . . 118
5.3.4. Resultados . . . 119
5.3.5. Comparación de M-ADAP con otras representaciones . . . 119
5.4. Conclusiones . . . 121

6. Conclusiones Generales **123**
6.1. Sumario . . . 123
6.2. Conclusiones . . . 124
6.3. Trabajos futuros . . . 127
6.4. Publicaciones . . . 128

A. Competición *RoboCup Coach 2006* **151**
A.1. Configuración de la Competición - *Modelado OffLine* . . . 151
A.1.1. Patrones de comportamiento - Ronda 1: . . . 151
A.1.2. Patrones de comportamiento - Ronda 2: . . . 153
A.2. Configuración de la Competición - *Detección OnLine* . . . 154
A.3. Análisis de los patrones . . . 157
A.4. Otros equipos participantes en la competición *Coach 2006* . . . 160
A.5. Resultados de la Competición *Coach 2006* . . . 161

B. Introducción a los Modelos Ocultos de Markov **163**
B.1. Definición de *Modelos Ocultos de Markov* . . . 163
B.2. Formalización de los HMM . . . 164
B.3. *Algoritmo Forward* . . . 165

C. Cálculo Recursivo del Potencial Utilizando Distancia Coseno **167**
C.1. Expresión Recursiva del Potencial (sin el cuadrado de la Distancia Coseno) . . . 168
C.2. Expresión Recursiva del Potencial (con el cuadrado de la Distancia Coseno) . . . 170

Índice de figuras

2.1. Construcción del HMM a partir de las observaciones del *mundo real* 21
2.2. Sistema TS Borroso representado como un sistema neuro-borroso. 31
2.3. Algoritmo utilizado por LVQ 34

3.1. Estructura general de M-COMP - Modelado de Agentes Mediante Comparación de Secuenciación de Eventos. 38
3.2. Ejemplo de *trie* utilizado para almacenar las palabras: «da», «dos», «don», «doy», «a» y «mis». 40
3.3. Pasos para la creación de un *trie*. 42
3.4. *Trie* que incluye el valor χ^2 en cada nodo. 44
3.5. Algoritmo comparación de *tries* para búsqueda de patrón de comportamiento 47
3.6. Algoritmo de comparación de *tries* para calcular su similitud . . . 48
3.7. BIBMOD-1 y BIBMOD-K creados a partir de una misma secuencia de eventos y utilizando una longitud de subsecuencia de tres elementos. 50
3.8. Esquema de la *Liga de Simulación 2D* de la *RoboCup*. 52
3.9. Monitor del sistema *Soccer Server*. Los agentes (jugadores) se representan como círculos y su color indica su equipo. La pelota se indica con el círculo blanco. Los agentes *coach* no están representados. 53
3.10. Estructura General de la competición *Coach*. 55
3.11. Análisis de Comportamientos - Detección online. *Coach 2006* - Ronda 1 64
3.12. Análisis de Comportamientos - Detección online. *Coach 2006* - Ronda 2 65
3.13. Resultados - Detección online. *Coach 2006* - Rondas 1 y 2 66
3.14. *Patrón_15*. Reglas *CLang* y descripción. 67

4.1. Estructura general de MAUSE - Modelado de Agentes Utilizando Secuenciación de Eventos. 70

4.2. *Trie* que incluye el valor *frecuencia relativa* en cada nodo. 71
4.3. Distribución de subsecuencias de eventos considerando su frecuencia relativa. 72
4.4. Funcionamiento de ABCD - Ejemplo 75
4.5. Aplicación de MAUSE en la competición *Coach* 76
4.6. Resultados aplicando MAUSE e utilizando métodos basados en Frecuencia. 78
4.7. Comandos Unix: Aplicación de MAUSE 83
4.8. MAUSE vs HMM - Porcentaje de aciertos en la clasificación. Validación cruzada con 10 particiones. Conjuntos de 9 Usuarios. . . 89
4.9. MAUSE vs HMM - Porcentaje de aciertos en la clasificación. Validación cruzada con 10 particiones. Conjuntos de 50 Usuarios. . . 90

5.1. Estructura general de M-ADAP: Modelado de Agentes Auto-Adaptativo con Flujo Continuo de Eventos. 92
5.2. Ejemplo para la representación de las distribuciones de subsecuencias de eventos en SAA 94
5.3. Interpretación del clasificador obtenido utilizando reglas borrosas. 101
5.4. Tasa de aciertos de la *nueva clase* (*Programadores Principiantes*) incrementando el número de usuarios pertenecientes a dicha clase en el conjunto de entrenamiento. 110
5.5. Tasa de aciertos de la *nueva clase* (*Programadores Expertos*) incrementando el número de usuarios pertenecientes a dicha clase en el conjunto de entrenamiento. 112
5.6. Tasa de aciertos de la *nueva clase* (*Científicos Informáticos*) incrementando el número de usuarios pertenecientes a dicha clase en el conjunto de entrenamiento. 112
5.7. Tasa de aciertos de la *nueva clase* (*No Programadores*) incrementando el número de usuarios pertenecientes a dicha clase en el conjunto de entrenamiento. 113
5.8. Plano del apartamento *inteligente* que incluye los sensores de movimiento utilizados en la toma de datos. 115
5.9. Valores obtenidos por los sensores utilizados en la realización de dos actividades cotidianas: *"Lavarse las manos"* y *"Cocinar"*. . . 117
5.10. Modelo de Markov que representa la actividad: *"Lavarse las manos"*. 121

B.1. Esquema de cadenas de Marvok y Modelos Ocultos de Markov (HMM) 164
B.2. *Algoritmo Forward* - Calcula la probabilidad de que una secuencia pertenezca a un modelo concreto. 166

Índice de tablas

3.1. Tabla de contingencia χ^2 43
3.2. Comportamientos utilizados en la ronda 1 y 2 de la competición *Coach 2006* 62
3.3. Umbrales utilizados por M-COMP en la competición *Coach 2006* 63

4.1. Resultados de aplicar MAUSE en el entorno propuesto basado en la competición *Coach 2006*. Ronda 1 y Ronda 2. 79
4.2. Comparación de resultados aplicando M-COMP y MAUSE en el dominio de competición *Coach 2006*. Ronda 1 y Ronda 2. 79
4.3. Porcentaje de aciertos en la clasificación utilizando MAUSE. Validación cruzada con 10 particiones. Conjuntos de 9 y 50 Usuarios. 85
4.4. Porcentaje de aciertos en la clasificación utilizando HMM. Validación cruzada con 10 particiones. Conjuntos de 9 y 50 Usuarios. 88

5.1. Descripción del conjunto de datos utilizado. 103
5.2. Número total de subsecuencias diferentes obtenidas. 105
5.3. M-ADAP: Número de prototipos creados por clase utilizando validación cruzada con 10 particiones 106
5.4. Porcentaje de aciertos de diferentes algoritmos en la clasificación de usuarios Unix utilizando diferentes números de comandos por usuario para el entrenamiento y validación (#1) y diferentes longitudes de subsecuencia (#2) 109
5.5. Ejemplo de secuencia de eventos generada por MCSE en la actividad *"Lavarse las manos"*. 118
5.6. Ejemplo de secuencia de eventos generada por MCSE en la actividad *"Cocinar"*. 118
5.7. Número total de subsecuencias diferentes generadas en función de la longitud de subsecuencia considerada 119
5.8. Porcentaje de aciertos de diferentes algoritmos de clasificación en la clasificación de actividades utilizando diferentes longitudes de subsecuencia (#) para la creación del modelo 120

A.1. Ejecución de los patrones en Iteración 1 - Ronda 1 de competición *Coach 2006* 154
A.2. Ejecución de los patrones en Iteraciones 2 y 3 - Ronda 1 de la competición *Coach 2006* 155
A.3. Ejecución de los patrones en la Ronda 2 - Iteraciones 1, 2 y 3 de la competición *Coach 2006*. 156
A.4. Análisis del tipo de acción del patrón - Ronda 1, Iteración 1 de la competición *Coach 2006* 157
A.5. Análisis del tipo de acción del patrón - Ronda 1, Iteraciones 2 y 3 de la competición *Coach 2006* 158
A.6. Análisis del tipo de acción del patrón - Ronda 2, Iteraciones 1, 2 y 3 de la competición *Coach 2006* 159
A.7. Resultados competición *Coach 2006*. Primera Ronda 161
A.8. Resultados competición *Coach 2006*. Segunda Ronda 162
A.9. Resultados competición *Coach 2006*. Tercera Ronda 162

Capítulo 1

Introducción

Las teorías más recientes sobre el cerebro humano confirman que un alto porcentaje de su capacidad es utilizado para predecir el futuro, incluyendo el comportamiento de otras personas [138]. Para actuar de la forma más adecuada en un contexto social, los humanos tratan de reconocer el comportamiento de las personas que les rodean y hacer predicciones basadas en estos reconocimientos.

Actualmente son muchos los entornos en los que reconocer el comportamiento de sus integrantes puede resultar de gran utilidad. Supóngase una residencia de discapacitados en la que se encuentran varios residentes. Cada residente se comporta de forma diferente en función de sus capacidades, sus hábitos o su estilo de vida. Si fuera posible crear perfiles de comportamiento de los distintos residentes, se podrían realizar tareas interesantes como detectar cambios en el comportamiento de un residente o agrupar comportamientos similares. Además, conocer el perfil de comportamiento de un residente permite poder facilitarle la ejecución de sus actividades rutinarias.

Por otra parte, la necesidad de desarrollar aplicaciones complejas compuestas de multitud de subsistemas que interaccionan entre sí, obliga a distribuir la inteligencia entre diversos agentes y a construir Sistemas Multiagente (SMA) [135]. En estos sistemas, el agente necesita, además del conocimiento del dominio, conocimiento sobre sus propias capacidades y conocimiento social para saber su papel en dicho dominio. La necesidad de adquirir información sobre otros agentes puede ser solventada mediante la comunicación. Sin embargo, esta comunicación entre agentes puede ser limitada, compleja y/o costosa. Además, en entornos competitivos, la comunicación con aquellos agentes con los que se compite, prácticamente no existe. Por este motivo, toda información obtenida sin necesidad de una comunicación explícita, es decir, inducida de la observación de otros agentes, es muy relevante en estos entornos.

En esta tesis doctoral se aborda la tarea de adquirir de forma automática conocimiento acerca del comportamiento de otros agentes inteligentes. El modelado de

agentes [110] es un proceso que permite a un agente extraer y representar conocimiento (comportamiento, creencias, metas, acciones, planes, etcétera) de otros agentes en un entorno determinado.

Los primeros trabajos relacionados con el modelado de agentes en entornos competitivos demostraron que adquirir información sobre cómo se comporta el oponente permite anticiparse a sus intenciones y, proporciona mejores resultados que reaccionar únicamente a sus últimas acciones [36]. Actualmente, un agente capaz de reconocer el comportamiento de otros agentes, puede realizar diversas tareas como predecir el comportamiento futuro de los agentes observados [177, 161], coordinarse con ellos [172], facilitarles la ejecución de sus acciones [159], detectar sus posibles errores [103] o detectar aquellos agentes considerados como intrusos [35].

La mayoría de técnicas actuales utilizadas para el modelado del comportamiento de agentes construyen una *biblioteca de comportamientos*, en la que se almacenan los modelos obtenidos. Esta tarea puede realizarse por un experto del dominio o de forma automática, siendo esta última la considerada en esta tesis doctoral. Cabe destacar que estas técnicas están en sus inicios y normalmente son dependientes del dominio.

Uno de los retos en el modelado de agentes es cómo afrontar este proceso cuando el comportamiento de los agentes varía constantemente. Por ejemplo, si los agentes que se desea modelar son humanos, se debe tener presente que su comportamiento normalmente cambia con el tiempo porque cambian sus metas, se olvidan antiguos hábitos, etcétera. Por lo tanto, los modelos de comportamiento creados deben ser revisados y actualizados cada cierto tiempo. Esto resulta un proceso complejo ya que es posible perder información reflejada en modelos previamente construidos.

Esta tesis doctoral propone tres enfoques diferentes para modelar y detectar el comportamiento de uno o varios agentes en distintos entornos. Estos enfoques permitirán que un agente situado en un entorno determinado, sea capaz de adquirir conocimiento acerca de otros agentes situados en el mismo entorno. Cada enfoque propuesto posee características particulares que le permiten adecuarse a un cierto tipo de dominios, lo que implica que se puede adquirir conocimiento sobre otros agentes en diversos SMA.

El primero de los enfoques propuestos en esta tesis doctoral, es capaz de identificar y modelar las diferencias en el comportamiento de un conjunto de agentes a partir de su observación. Este enfoque se centra en la resolución de la problemática propuesta en la competición *RoboCup Coach 2006* [184]. Esta competición se centra en la detección de patrones de comportamiento del equipo oponente en un entorno de fútbol simulado.

El segundo enfoque propuesto tiene como finalidad modelar y clasificar el

comportamiento de uno o varios agentes en un determinado entorno. A diferencia del enfoque anterior, éste no necesita comparar comportamientos, sino que el modelo se obtiene a partir de la observación de un único comportamiento de uno o varios agentes. Una vez creados los modelos, se utiliza un método estadístico para clasificar el comportamiento de un agente observado en el mismo dominio. La generalidad de este enfoque permite que pueda ser evaluado en dos entornos muy diferentes tanto en sus propiedades como en su finalidad: la categorización de equipos de fútbol simulado y la identificación de usuarios de una interfaz de línea de comandos.

Por último, el tercer enfoque propuesto en esta tesis doctoral propone modelar el comportamiento de uno o varios agentes con la particularidad de que dicho modelo pueda ser actualizado en función de nuevas observaciones de los agentes. Este enfoque se basa en los Sistemas Auto-Adaptativos (SAA) [15], capaces de modificar tanto sus parámetros como su estructura en función de los datos recibidos. La evaluación de este enfoque se ha realizado en tareas de modelado y clasificación de comportamientos de usuarios de una interfaz de comandos y en el reconocimiento de actividades en un entorno inteligente.

1.1. Objetivos

El objetivo general de esta tesis doctoral consiste en *definir un marco que permita crear modelos de comportamiento de agentes inteligentes de forma automática a partir de la observación de su comportamiento, pudiendo además detectar el comportamiento de un agente observado posteriormente.*

Debido a que todas las acciones ejecutadas por un agente se realizan de forma secuencial, en este trabajo se considerará que todo comportamiento puede representarse por una secuencia de acciones o eventos. Cómo se obtiene dicha secuencia a partir de la observación es dependiente del dominio y del tipo de comportamiento que se desea modelar.

Para lograr este objetivo general, se deben cumplir unos objetivos específicos que se citan a continuación:

- Definición de un método que genere una secuencia de eventos a partir de la observación de un comportamiento. Dado que todo comportamiento debe ser representado como una secuencia de eventos, se deberá desarrollar un método capaz de generar dichas secuencias en cada uno de los dominios utilizados en esta tesis doctoral. Este método es dependiente del dominio, por lo que la implementación del marco propuesto en un nuevo dominio implica definir este método ajustándose a las características del dominio. Para ello, se deben definir los eventos que se desean reconocer a partir de

las observaciones del comportamiento de uno o varios agentes en dicho dominio.

- Desarrollo de un método capaz de modelar el comportamiento de un agente en un entorno determinado. Se debe desarrollar un método que permita definir y representar el comportamiento de uno o varios agentes. Dado que un modelo de comportamiento es una aproximación simplificada de la realidad, la representación creada deberá almacenar toda información relevante relacionada con el comportamiento o comportamientos observados.

- Desarrollo de un método que permita llevar a cabo el almacenamiento de los modelos creados. Todo modelo creado deberá poder ser almacenado para su posterior utilización. Para ello, es necesario el desarrollo de un método capaz de crear y utilizar una biblioteca de modelos de comportamiento.

- Desarrollo de un método capaz de identificar el comportamiento de un agente en un entorno observado. Una vez almacenados una serie de modelos en su correspondiente biblioteca, se debe implementar un algoritmo capaz de clasificar un nuevo comportamiento observado como aquél al que más se asemeje de los almacenados en la biblioteca de modelos.

Dado que el modelado del comportamiento de agentes resulta útil en muchas y muy diversas aplicaciones, resulta difícil abordar todas estas aplicaciones desde una misma perspectiva. Por este motivo, en esta tesis doctoral se diferencian tres tipos de dominios:

1. Dominios en los que se dispone de dos comportamientos diferentes de un mismo agente o conjunto de agentes. En este caso, se quiere detectar y modelar la diferencia entre ambos comportamientos.

2. Dominios en los que el modelo de uno o varios agentes debe ser creado a partir de la observación de un único comportamiento. Una vez modelado el comportamiento de un agente, éste no varía considerablemente.

3. Dominios en los cuales, los agentes modifican su comportamiento a lo largo del tiempo y en donde estos cambios tienen que reflejarse en el modelo de comportamiento creado. Esta actualización necesita de una observación continua de los agentes. Además, en estos dominios el comportamiento de un agente es clasificado como un comportamiento general y no particular.

Considerando estos tres tipos de dominios, esta tesis doctoral define tres marcos conceptuales diferentes en función de las características particulares de cada uno de ellos. La diversidad de estos marcos hace que el objetivo planteado pueda llevarse a cabo en SMA muy distintos.

1.2. Organización de la Memoria

Con la finalidad de dar una visión general de los temas que se tratan en esta tesis doctoral, en el capítulo 2 se revisa la literatura disponible sobre el modelado de agentes. Además, y dado que los enfoques propuestos se basan en el análisis de secuencias, también se incluye un estudio sobre los trabajos relacionados con el aprendizaje en secuencias de elementos. Por último, en este capítulo se realiza una introducción a los Sistemas Auto-Adaptativos (*Evolving Systems*) por su utilización en esta tesis doctoral.

En el capítulo 3 se presenta el primero de los enfoques propuestos en esta tesis doctoral. Este enfoque está basado en el modelado del comportamiento de un conjunto de agentes a partir de la observación de dos comportamientos diferentes de un mismo agente o conjunto de agentes. Se ha denominado M-COMP y está orientado a resolver la problemática planteada en la competición *RoboCup Coach 2006*.

En el capítulo 4 se presenta MAUSE, el segundo de los enfoques propuestos. MAUSE se centra en el modelado y detección del comportamiento de uno o varios agentes a partir de la observación de un único comportamiento.

En el capítulo 5 se propone y evalúa el tercero de los enfoques de esta tesis doctoral y que se ha denominado M-ADAP. Este enfoque considera el modelado de agentes a partir de un flujo continuo de eventos. Este tipo de modelado se denomina adaptativo porque está en continua actualización en función de nuevas observaciones.

En el capítulo 6 se detallan las conclusiones generales, las aportaciones de esta tesis doctoral y las líneas de trabajo futuras.

Capítulo 2

Estado del Arte

En este capítulo se presentan las investigaciones más relevantes relacionadas con los diversos temas que involucra esta tesis doctoral. El apartado 2.1 revisa las diferentes definiciones de Agentes y Sistemas Multiagente (SMA). Las investigaciones más relevantes realizadas en el modelado de agentes se resumen en el apartado 2.2. Dado que para el modelado de agentes en este trabajo se analizan secuencias de eventos que representan comportamientos, el apartado 2.3 presenta el concepto de secuencia utilizado en esta tesis y expone diversas investigaciones sobre aprendizaje en secuencias de elementos. Por último, en el apartado 2.4 se explicará qué son los Sistemas Auto-Adaptativos (Evolving Systems) y por qué son de utilidad en esta tesis doctoral.

2.1. Agentes y Sistemas Multiagente

2.1.1. El concepto de Agente

La creciente investigación en el campo de la Inteligencia Artificial (IA), y más concretamente en la Inteligencia Artificial Distribuida (IAD) ha llevado a afrontar problemas complejos a través de las ventajas que supone la utilización de agentes [135].

Uno de los primeros acercamientos al término agente fue propuesto en el año 1982 por Newell [142] que consideraba que en el diseño de sistemas existían dos niveles de abstracción: el nivel de símbolos y el nivel de conocimiento. En el nivel de símbolos, se contemplan aquellos programas convencionales que sólo manejan símbolos (expresiones o variables). Por otra parte, en el nivel de conocimiento, un agente, equivalente a un programa en este nivel, maneja conocimiento. De esta forma, un sistema en el nivel de conocimiento de Newell puede considerarse un agente. Así, este agente está formado por una estructura, unas acciones que éste

puede realizar y unas metas que desea alcanzar, y además se comporta de acuerdo con el *principio de racionalidad* que determina qué acciones debe realizar para alcanzar sus metas.

Actualmente, existen muchas definiciones del término agente [57, 66, 68, 190] aunque ninguna de ellas está totalmente aceptada por toda la comunidad investigadora. Una de las más citadas es la propuesta por Wooldridge [191] en el año 1997: *"Un agente es un sistema informático situado en un entorno y que es capaz de realizar acciones de forma autónoma para conseguir sus objetivos de diseño"*. De acuerdo con ésta y otras definiciones [93, 190], la característica principal de un agente podría ser su autonomía [40]. El resto de cualidades que debe tener un agente pueden depender del entorno en donde se defina el concepto [93].

En general, un agente puede definirse como un sistema informático, situado en un entorno, dentro del cual actúa de forma autónoma y flexible para cumplir unos objetivos. Así, un agente recibe entradas de su entorno y a la vez ejecuta acciones para intentar modificarlo en función de sus metas. Además de la interacción con el medio, Wooldridge y Jennings [190] citan una serie de características básicas que debe cumplir un agente:

- Sociabilidad: Capacidad de interaccionar con otros agentes por medio de un lenguaje de comunicación.
- Reactividad: Capacidad para percibir estímulos del entorno y reaccionar ante ellos.
- Iniciativa: Capacidad de tomar la iniciativa para actuar guiado por los objetivos a satisfacer.

2.1.2. Sistemas Multiagente

Dado el carácter distribuido que tienen los entornos en los que se aplican agentes, resulta adecuado considerar grupos de agentes que interaccionan entre sí para conseguir objetivos comunes. Según la *Object Management Group* [1], un SMA puede definirse como: *Una plataforma que puede crear, interpretar, ejecutar, transferir y terminar agentes*.

Los SMA son sistemas computacionales en los que varios agentes autónomos interaccionan entre sí; ya sea para colaborar en la solución de un conjunto de problemas o en la consecución de una serie de objetivos individuales o colectivos [51]. Estos agentes pueden ser homogéneos o heterogéneos, pueden tener

[1] consorcio dedicado al cuidado y el establecimiento de diversos estándares de tecnologías orientadas a objetos. Web: http://www.omg.org

metas comunes o no y poseen, en la mayoría de los casos, algún grado de comunicación entre ellos. Cada uno de estos agentes podría tener comunicación directa con seres humanos a través de interfaces o agentes de usuario.

En [173] se destacan las siguientes características de los SMA:

- Cada agente tiene información incompleta o un punto de vista limitado para resolver el problema.
- No existe un control global para todo el sistema.
- Los datos están descentralizados
- La computación es asíncrona

Por otro lado, tal y como se indica en [173], los SMA se caracterizan por su modularidad. Si un dominio es particularmente complejo, amplio o impredecible, éste puede ser simplificado mediante el desarrollo de un número de componentes (agentes) funcionales, específicos y modulares. Esta descomposición permite a cada agente usar el paradigma más apropiado para resolver su correspondiente problema. Sin embargo, debido a esta división, todo agente debe ser capaz de coordinarse con otros para resolver problemas de interdependencia.

La tecnología de SMA está realizando importantes aportaciones en la resolución de problemas en diversos dominios donde aproximaciones tradicionales no proporcionan soluciones suficientemente satisfactorias; comercio electrónico [34, 35], telemedicina [18], vídeo vigilancia [72] y un largo etcétera [182].

2.2. Modelado de Agentes

El término *modelado de agentes* se podría definir como *la capacidad de adquirir, inferir y almacenar el conocimiento (comportamiento, creencias, metas, acciones, planes...) de otros agentes* [110]. Así, la finalidad principal de todo modelado de agentes es inferir, a partir de la observación, qué está haciendo o intentando hacer un agente [51]. Como ejemplo, se puede considerar cómo de la observación de un jugador de fútbol, se puede extraer su posición y su velocidad y con estos datos inferir por ejemplo, si un determinado jugador intentó pasar la pelota y a quién, si éste quiere ganar o perder el partido, o incluso el estado de ánimo de dicho jugador.

El estudio de la creación de agentes que sean capaces de obtener conocimiento de otros y así poder actuar en consecuencia, ha sido y sigue siendo de interés en una gran cantidad de campos.

En dominios colaborativos, se requiere una coordinación de las actividades de los agentes para que éstos puedan cooperar. Para ello, es necesario establecer

mecanismos de comunicación basados en un lenguaje que permita el intercambio de conocimiento [75]. Sin embargo, también se puede obtener dicha información a partir de la observación de los otros agentes. Es decir, las observaciones obtenidas pueden ser analizadas para inferir la información necesaria y evitar así una comunicación explícita [178]. En este caso, el modelado de agentes supone una ausencia de comunicación que conlleva, entre otras ventajas, un ahorro de recursos.

Por otra parte, en dominios competitivos, modelar otros agentes para obtener datos acerca de su comportamiento, resulta de gran ayuda a la hora de predecir futuros comportamientos. Estas predicciones pueden utilizarse en la toma de decisiones para crear planes adaptativos a su propio juego o al del oponente [117].

El modelado de otros agentes es de gran utilidad en tareas como la ayuda o asistencia a personas en un entorno inteligente [48], en la imitación de las acciones de los agentes adversarios [19] o en la detección de fallos en un SMA [103, 105]. Por esta razón, son muchas las aplicaciones en las que se emplea el modelado de agentes, entre ellas, se pueden destacar:

- Aplicaciones que incluyen diálogos en lenguaje natural (comprensión del lenguaje y generación de respuestas) [8, 32, 86] y sistemas de traducción de lenguaje [7].
- Aplicaciones para el desarrollo de interfaces de usuario inteligentes [80].
- Aplicaciones de resolución de problemas en colaboración [128] y coordinación multiagente [102, 105].

Las investigaciones en el área de modelado de agentes se han desarrollado en gran variedad de dominios y se han utilizado técnicas diferentes. En las siguientes secciones se detallan las formas utilizadas más relevantes para afrontar este problema.

2.2.1. Teoría de Juegos y el Modelado de Oponentes

La teoría de juegos [141] es un enfoque que examina el comportamiento estratégico para toma de decisiones en situaciones de conflictos, y por lo tanto la base del modelado de agentes; en este caso, el agente que se desea modelar es el propio oponente por lo que este proceso se conoce como *modelado de oponentes*.

La teoría de juegos clásica hace referencia a un juego finito en el que hay una serie de jugadores que pueden realizar un conjunto de acciones o movimientos discretos bien definidos. En cada turno, el jugador tiene que seleccionar una acción de un conjunto de acciones posibles. Cada jugador conoce, no solo las acciones

que puede realizar él, sino también las posibles acciones del oponente. El conjunto de acciones que el jugador selecciona, dependerá de su estrategia.

Conocer la estrategia del oponente puede ser de gran ayuda para planificar la de los agentes a los que se indica cómo actuar. A partir de esta consideración, comenzaron las primeras investigaciones sobre técnicas que permitieran obtener un modelo de comportamiento del oponente. Así, el modelado del oponente fue aplicado en juegos clásicos como el ajedrez [165].

En la teoría de juegos, el algoritmo *minimax* [165] es uno de los procedimientos más conocidos y utilizados para problemas con exactamente dos adversarios, movimientos alternos e información sin ruido. En este caso, las acciones que se realizan dependen de la estrategia del oponente, pero siempre se considera que éste juega de forma óptima. Sin embargo, en entornos complejos, el oponente normalmente juega de forma sub-óptima [99], ya que encontrar la mejor estrategia puede no ser posible. Para optimizar este algoritmo, se puede limitar la búsqueda por nivel de profundidad, por tiempo de ejecución o la conocida como *poda alfa-beta* que considera que el jugador oponente no permitirá realizar las mejores jugadas de su contrincante.

A pesar de estas mejoras, en muchos entornos intervienen más de dos jugadores por lo que Korf [114] considera no sólo la posibilidad de utilizar diferentes funciones de evaluación en diferentes jugadores, sino también permitir un número arbitrario de jugadores.

Además, el algoritmo *minimax* también asume que el jugador no tiene conocimiento sobre el procedimiento de toma de decisión del oponente. Sin embargo, es posible acceder a cierto conocimiento sobre el oponente a partir de su observación durante un determinado número de iteraciones.

Este conocimiento, que puede ser adquirido a partir de la observación del comportamiento del oponente, puede ser utilizado considerando dos enfoques:

- La búsqueda de cómo realiza la toma de decisiones el oponente con la finalidad de realizar la *mejor* acción.
- El empleo de métodos capaces de inferir y aprender el modelo del oponente.

Cada uno de estos dos enfoques se detalla en los siguientes sub-apartados.

Búsqueda de cómo realiza la toma de decisiones el oponente

El juego especulativo propuesto por Jensen en [100] está muy relacionado con la búsqueda de cómo se comporta el oponente utilizando para ello su modelo. Jensen propone una técnica en la que se selecciona el *mejor* movimiento entre los óptimos, considerando mejor al que ofrece más oportunidades para la confusión del oponente. Para ello, se utiliza una técnica denominada *ProbiMax* capaz de

modelar al oponente implícitamente por sus probabilidades de movimiento. Este enfoque se evalúa en el juego del ajedrez.

Lida et al. [130] proponen en el año 1993 buscar cómo se comporta el oponente en su modelo representado por un árbol de juego (*Opponent-Model search*). Uno de los objetivos de este enfoque es poder considerar diferencias en la función de evaluación de los jugadores. Si un oponente está utilizando una función de evaluación débil, el jugador podría detectarlo y jugar con ventaja.

El enfoque anterior es mejorado en 1996 por Carmel y Markovitch [36] quienes presentan un algoritmo denominado *M** que utiliza una estructura recursiva para obtener información sobre cómo razona el oponente sobre otros jugadores. Así, antes de ejecutar una acción, el algoritmo planteado simula los estados sucesores al actual y aplica la estrategia propia del modelo del oponente sobre cada estado resultante simulando así la respuesta que éste podría dar. Finalmente, cada respuesta considerada se evalúa aplicando el algoritmo de forma recursiva. Además, se introduce la utilización de clases de oponentes por lo que los modelos no son construidos en cada juego, sino que se considera un conjunto fijo de modelos que abarca todos los posibles oponentes. En este caso, el juego contra un oponente implica primero su clasificación para poder así seleccionar la acción más apropiada. Sin embargo, esta propuesta es aplicable únicamente en juegos por turnos y no se puede extrapolar a dominios complejos con varios agentes. Además, en dominios con acciones no discretas, comprobar secuencialmente todos los estados sucesores posibles, no es factible. Por último, también hay que destacar que considerando las propiedades anteriores, resulta muy complicado crear una función de evaluación eficiente y acertada.

Aprendiendo el Modelo del Oponente

Con la finalidad de inferir el modelo de comportamiento del oponente, el comportamiento que se observa de éste puede representarse de muy diversas formas. A continuación se exponen las representaciones más relevantes:

En el año 1996, Carmel y Markovitch [37] presentan un método para construir un modelo del oponente consistente con las entradas y salidas observadas de dicho oponente. La interacción entre agentes se representa en dicho método como un juego de dos jugadores donde el objetivo es buscar una estrategia que maximice la suma de las recompensas esperadas en el juego. El método utilizado se basa en las entradas y salidas del oponente (comportamiento). La estrategia se modela mediante un autómata finito determinista, AFD, por lo que éste se puede calcular en tiempo polinómico [148].

Teniendo en cuenta la suposición anterior que considera que una estrategia se puede modelar como un AFD, sería posible obtener el AFD mínimo que defina el comportamiento de dicho oponente. Sin embargo, encontrar este AFD míni-

mo es un problema NP-completo [79] por lo que Carmel y Markovitch proponen un algoritmo denominado *US-L** (*unsupervised L**). Este algoritmo es capaz de aprender el modelo de forma no supervisada, manteniendo una tabla de observaciones, ya que realizar experimentos con el oponente podría ser muy costoso o imposible. El algoritmo implementado trata en todo momento de mantener un modelo consistente en lugar de un modelo óptimo. Para ello, si una observación es inconsistente con el modelo, dicho modelo se modifica.

2.2.2. Reconocimiento de planes

Kautz y Allen [108] definen el reconocimiento de planes como el problema de identificar un conjunto mínimo de acciones de alto nivel suficientes para explicar las acciones observadas. Este problema está relacionado con la planificación tradicional en Inteligencia Artificial, de forma que el reconocimiento de planes es una planificación a la inversa. La planificación está basada en la generación de una serie de planes que, a partir de la ejecución de unas acciones, son capaces de alcanzar una meta. Por otra parte, el reconocimiento de planes intenta inferir el plan que ejecuta un agente a partir de la observación de sus acciones. Por este motivo, el reconocimiento de planes implica, inherentemente, más incertidumbre que la planificación.

Según Cohen [47], un sistema de reconocimiento de planes puede ser de dos tipos: *keyhole* (ojo de cerradura) o intencionado. En un sistema de reconocimiento de planes *keyhole*, los agentes observados no se ven afectados por el proceso de reconocimiento. En los sistemas intencionados, los agentes observados realizan acciones deliberadamente que puedan ayudar a dicho reconocimiento. Además, Geib y Goldman [73] proponen otra clase denominada reconocimiento de adversario (*adversarial recognition*), en el que el agente observado es hostil y tratará de confundir o engañar al observador.

Una de las investigaciones pioneras en este campo fue desarrollada en 1978 por Schmidt et al. [162] y en ella se detallaba un sistema denominado *BELIEVER* basado en teorías psicológicas para entender cómo los seres humanos inferían los planes de otros mediante la observación de sus acciones.

En la década de 1980, se consideraron tres enfoques diferentes en el reconocimiento de planes:

- El primero de los enfoques está basado en la explicación [43] y consiste en explicar un conjunto de observaciones basándose en las suposiciones proporcionadas por las observaciones. El principal problema es la gran cantidad de suposiciones que deben considerarse y la dificultad para seleccionar una de ellas.

- Otro de los enfoques está basado en el análisis sintáctico [92, 166] en el que se consideran las acciones como secuencias de subacciones y el comportamiento se modela como reglas libres de contexto en una *gramática de acciones*. Así, las observaciones se tratan como entradas para el análisis sintáctico y se obtiene el árbol sintáctico que las explique.

- El último de los enfoques considerado está basado en el concepto de inferencia similar [9]. En estos sistemas, un conjunto de reglas se utiliza de la forma: *Si se observa la acción A, ésta podría ser una parte de la acción B*. Estas reglas determinan un espacio de búsqueda de acciones que producen planes que incluye las observaciones. Sin embargo, el control de la búsqueda se encuentra oculto en un conjunto de heurísticas por lo que definir un plan de forma precisa es una tarea compleja.

Bibliotecas de Planes

Independientemente del enfoque utilizado, todo sistema de reconocimiento de planes debe ser capaz de representar los diferentes planes reconocidos de alguna manera para poder almacenarlos en lo que se conoce como biblioteca de planes (*plan library*). Son muchas las representaciones de planes propuestas, como por ejemplo, taxonomía de acciones [108], red de tareas jerárquica [63], redes bayesianas [42], etcétera.

El trabajo de Kautz y Allen [108] publicado en 1986, propone una teoría formal del reconocimiento de planes significativamente más adecuada que las anteriores, y que no tiene restricciones en las relaciones temporales entre las acciones. En dicha investigación, el reconocimiento de planes se trata de forma deductiva y considera una representación denominada *taxonomía de acciones* que puede definirse como una descripción exhaustiva de la forma en que las acciones se ejecutan. Es decir, en la *taxonomía de acciones*, todas las acciones observadas son una parte de uno o más *planes de alto nivel*. La principal tarea del reconocimiento de planes consiste en encontrar el conjunto mínimo de los planes de alto nivel que expliquen dichas observaciones.

Tal y como se explica en [17], este enfoque propuesto por Kautz y Allen se encuentra con grandes dificultades cuando se aplica en entornos complejos y dinámicos como los SMA. La primera de ellas es que las acciones de los agentes podrían ser continuas, y modelar tales acciones usando operadores discretos conlleva gran complejidad. El segundo problema radica en que la entrada al algoritmo de reconocimiento de planes es una cadena de acciones, de forma que es difícil incorporar contexto adicional que pueda ser observable y que pueda afectar a la toma de decisión interna del agente observado. Por último, en el enfoque de Kautz tampoco se considera que, dado que los agentes operan en entornos complejos

y dinámicos, éstos podrían interrumpir su secuencia de acciones planeada para reaccionar ante situaciones no esperadas. Este último problema es planteado y analizado en diversos trabajos [20, 143].

En 1995, Tambe et al. [177] presentaron un interesante algoritmo para el *seguimiento de agentes* en entornos flexibles y reactivos (*RESC: Real-Time Situated Commitments*) que ha sido referente en el reconocimiento de planes. Con el fin de inferir el comportamiento de un agente observado, *RESC* utiliza una representación basada en un comportamiento jerárquico. Mediante *RESC* un agente puede llevar a cabo el seguimiento de otro infiriendo una jerarquía de operadores del agente modelado aprovechando su propia arquitectura. *RESC* se basa en la habilidad de observar las acciones de los otros agentes en el estado actual del entorno. A pesar de las ambigüedades que esto conlleva, *RESC* considera una determinada interpretación sobre las acciones de los otros agentes rápidamente, sin realizar un análisis exhaustivo de todas las posibles alternativas. Así, *RESC* ejecuta el modelo del otro agente y compara sus acciones con las predicciones generadas por el modelo para verificar posibles fallos. En cada ciclo de ejecución, *RESC* mantiene únicamente una hipótesis sobre el estado actual del agente observado. Si después de la elección de un plan, se tuviera información adicional disponible, las interpretaciones pueden revisarse en tiempo real siendo capaz de retroceder para recuperarse de fallos ocurridos por ambigüedades. *RESC* puede utilizarse en tiempo real por su buena adecuación al entorno y a una recuperación de fallos inmediata. Este trabajo es ampliado por Tambe [178] y se centra en el uso del reconocimiento de planes ejecutados por equipos de agentes, y no por agentes individuales. El modelo utilizado se denomina $RESC_{team}$ y tiene en cuenta la formación de equipos y la asignación de roles.

En el año 2000, Kaminka y Tambe [103] proponen un algoritmo basado en *RESC* pero que mantiene múltiples hipótesis para el estado actual. Dicho algoritmo se denomina *RESL* (*Real-time Situated Least-commitment*) y se reajusta con cada nueva observación sin tener en cuenta el historial de las observaciones.

En resumen, el reconocimiento de planes se enfoca en agentes deliberativos que siguen una serie de planes jerárquicos con acciones que tienen un determinado orden. Estas condiciones implican que los comportamientos reactivos o planes que cambian de forma rápida, no pueden ser modelados. Además, a pesar de que la mayoría de los sistemas de reconocimiento de planes solamente tienen en cuenta las acciones realizadas por los agentes, hay otros muchos aspectos como el estado del mundo o las creencias de los agentes que pueden influir en el proceso de reconocimiento. Por último, se debe considerar que varios de estos sistemas no pueden utilizarse en entornos reales porque todas las observaciones necesarias, pueden no ser totalmente accesibles.

Enfoques probabilísticos

Los enfoques basados en modelos formales de probabilidad han tenido mucha importancia durante la década de 1990. Tras el trabajo de Kautz y Allen mencionado en el apartado anterior, Carberry [31] utiliza la teoría matemática *Dempster-Shafer* para el reconocimiento de planes ya que considera que un modelo no es adecuado si no incorpora cierta incertidumbre. Con la misma finalidad, Charniak y Goldman [42] proponen un enfoque probabilístico en el que utilizan Redes Bayesianas [44] en las que se insertan las explicaciones candidatas para el reconocimiento del plan. En estas Redes Bayesianas, los nodos *raiz* representan las hipótesis sobre los planes del agente y las variables aleatorias (nodos) representan las proposiciones. Cada probabilidad del nodo indica la probabilidad de la proposición dada la evidencia de sus *padres* y su *hijo*. Cuando se añade una evidencia a la red, las probabilidades de cada nodo se calculan de nuevo propagando las evidencias sobre los nodos. Sin embargo, este enfoque presenta una serie de problemas. Por ejemplo, requiere una gran cantidad de probabilidades accesibles, no hay distinción entre planes y acciones, y no tiene en cuenta el orden de los planes.

Por otra parte, Albrecht et al. [5] proponen un sistema de inferencia de planes *keyhole* basado en una red de creencia dinámica (*Dynamic Belief Network*) que representa las características del dominio necesarias para identificar los planes y metas de los agentes. Las distribuciones de probabilidad condicional para cada red se aprenden durante una fase de entrenamiento que construye dinámicamente estas probabilidades a partir de las observaciones del comportamiento de los agentes. Los resultados de este trabajo determinan que el enfoque propuesto es adecuado en entornos con suficientes datos de entrenamiento y cuya estructura causal de la red es claramente identificable.

Para representar el proceso de generación de planes de los agentes, Pynadath et al. [153] introducen la representación de gramáticas probabilísticas dependientes del estado (*PSDGs: Probabilistic State-Dependent Grammars*). Por su parte, Geib et al. [74] proponen un reconocedor de planes probabilístico denominado *PHATT* (*Probabilistic Hostile Agent Task Tracker*) que se utiliza en entornos en los que no pueden observarse todas las acciones que realizan los agentes.

2.2.3. Modelado de agentes en Sistemas Multiagente

El modelado de agentes en SMA parte de las investigaciones ya realizadas en la teoría de juegos y el reconocimiento de planes clásico. Sin embargo, este modelado tiene que hacer frente a situaciones más complejas. Esta complejidad surge porque en estos sistemas se expresan objetivos comunes y se realizan acciones en las que intervienen varios agentes. La búsqueda de una estrategia óptima

para un agente puede ser un problema complicado porque depende también del comportamiento de los otros agentes involucrados.

En uno de los primeros trabajos sobre este tema, Devaney y Ram [56] realizan una combinación de detección de patrones, planes y seguimiento de objetos para el reconocimiento de tácticas militares durante batallas de entrenamiento. Sin embargo, a pesar de estar enfocado en un contexto multiagente, no explota las nociones explícitas del trabajo en equipo.

Intille y Bobick [96] presentan un marco probabilístico para reconocer, mediante la observación, jugadas de fútbol americano en las que intervienen varios jugadores. Con la técnica propuesta, se crean modelos de eventos concretos y además, se definen relaciones temporales entre estos eventos. Por ejemplo, el evento *correr-con-el-balón* podría tener como requisito que la acción *coger-balón* ocurra *antes*.

Por su parte, Kaminka et al. [105] tratan el problema de monitorización de múltiples agentes basándose en el reconocimiento de planes. Para ello, utilizan un modelo de equipo avanzado que puede, por ejemplo, predecir fallos de estados y recuperar estados anteriores. También es capaz de detectar el cambio de rol en un equipo y continuar con su monitorización. Este tratamiento del equipo lo realiza utilizando un número muy limitado de observaciones de determinados agentes en determinados instantes. Esto lo diferencia de los trabajos comentados anteriormente de Devaney y Ram [56], y de Intille y Bobick [96] en donde se utilizan observaciones de todos los agentes en todos los instantes de tiempo.

Lerman y Galstyan [127] presentan una técnica para crear automáticamente modelos matemáticos del comportamiento colectivo en un SMA. El proceso para construir el modelo tiene 4 pasos:

1. Observar el comportamiento de los agentes y obtener sus acciones discretas.

2. Inducir un modelo del comportamiento de los agentes considerando las observaciones.

3. Convertir el modelo en un conjunto de ecuaciones diferenciales que describan el comportamiento de grupo.

4. Analizar las ecuaciones para aprender más sobre el sistema.

Esta metodología se ha aplicado en búsqueda, colaboración y asignación de tareas en grupos de agentes.

Relacionado con el modelado de comportamiento, Pérez et al. [151] utilizan Modelos Ocultos de Markov (HMM del inglés *Hidden Markov Models*) para el reconocimiento de actividades humanas. En este caso, se centran en el modelado de datos y la clasificación de acciones mediante el análisis de las características

extraídas de las imágenes proporcionadas por una cámara de vídeo. En un trabajo muy relacionado, con la finalidad también de reconocer actividades de humanos mediante imágenes de vídeo, Piccardi y Pérez [150] presentan una modificación de los HMM con probabilidades que se modelan utilizando el método de ventana *Parzen* (también conocido como *Kernel density estimation*).

Recientemente, Gal y Pfeffer [71] presentaron un lenguaje compacto y altamente expresivo para el razonamiento sobre las creencias de los agentes y los procesos de toma de decisiones. Este lenguaje se conoce como redes de diagramas de influencia (*Networks of Influence Diagrams, NID*) y representa un modelo de un agente. *NID* son estructuras gráficas en las que los modelos mentales de los agentes se representan como nodos de una red. De esta forma, un modelo mental para un agente podría por sí mismo utilizar las descripciones de los modelos mentales de otros agentes. Estos modelos pueden utilizarse para describir agentes que se comportan de forma irracional, es decir, que su comportamiento no se corresponde con la mejor respuesta, dadas las creencias sobre el *mundo* y los otros agentes. Además, este lenguaje puede utilizarse para aprender el comportamiento de otros agentes y sus procesos de razonamiento.

Mediante la utilización de HMM en SMA, Bui et al. [28] presentan un método para el reconocimiento de comportamientos de agentes en entornos dinámicos, con ruido y múltiples niveles de abstracción. Este método se evalúa en el reconocimiento de comportamientos utilizando los datos en tiempo real de las cámaras de vídeo de un sistema de vigilancia. En un entorno muy similar y basándose en este trabajo, Saria et al. [161] presentan un marco teórico para el reconocimiento de planes probabilísticos en SMA cooperativos. Este marco utiliza una Red Bayesiana Dinámica que permite razonar sobre la interacción entre múltiples agentes cooperativos.

Modelado del oponente en la *RoboCup*

Quizás uno de los dominios de prueba más utilizados para este tipo de investigaciones, y que será utilizado también para evaluar dos de los enfoques propuestos en esta tesis doctoral, es el simulador de fútbol de la RoboCup [110]. Tanto la competición como el simulador, se detallan en el apartado 3.6.

En este dominio, Stone et al. [172] proponen la clasificación del adversario en clases predefinidas como principal enfoque para el modelado de oponentes en SMA. Esta clasificación se realiza teniendo en cuenta dominios complejos (aquellos con gran cantidad de estados, incluso continuos, y acciones), como el proporcionado por *RoboCup*. El modelo propuesto se denomina *IMBBOP* (*Ideal Model Based Behavior Outcome Prediction*) y se basa en predecir el comportamiento de un agente en relación al comportamiento ideal de dicho agente en una situación dada.

Riley y Veloso [157] proponen una serie de pasos para modelar el comportamiento de agentes:

1. Extracción de las características necesarias del estado del mundo en el que se encuentran los agentes. Esta extracción de características dependerá del entorno que se esté tratando.

2. Construcción de las clases de adversario a partir de la información contenida en las características extraídas. Estas clases del adversario deberán plasmar la información estratégica sobre cómo se comporta cada tipo de adversario.

3. Comparación de las observaciones del adversario actual con las clases de adversario predefinidas, aportando así una clasificación de los comportamientos del adversario. Para entornos multiagentes y dinámicos, se establece una métrica que realiza esta comparación.

4. Realización de cambios en el comportamiento de los agentes basado en la clasificación realizada. Para esto, se deben considerar las estrategias que pueden desarrollar los agentes dependiendo de las clases de adversarios establecidas.

Por otra parte, Steffens [174] propone un modelado de oponentes basándose en el estudio de características realizado por Webb y Kuzmycz [185]. Así, en vez de describir el comportamiento del oponente completo, como se considera en otros enfoques, este modelo, denominado *FBDOM* (*Feature-Based Declarative Opponent-Modelling*), se enfoca en identificar los movimientos tácticos del oponente. De la misma manera que los humanos crean sus propios modelos comparando las nuevas observaciones con aquellas que fueron observadas y analizadas en el pasado, el comportamiento de los agentes se compara con los modelos ya almacenados de los oponentes.

Riley y Veloso [158] presentan un enfoque denominado *ATAC* (*Adaptive Team-Adversarial Coaching*) que se aplica en este dominio porque permite utilizar un agente especial denominado *coach* (entrenador) que es capaz de obtener información sin ruido sobre todos los agentes que participan en el partido. En este caso, el agente *coach* dispone de una serie de modelos de equipos oponentes previamente definidos y que están representados por probabilidades de posiciones de los jugadores. Además, el agente también es capaz de generar planes *on-line* en forma de Redes Temporales Simples [55] basándose en el reconocimiento de los planes del oponente. Por último, estos planes se comunican a los agentes que forman el equipo para que puedan utilizarlos y mejorar así su comportamiento.

En este mismo dominio, Ledezma et al. [126] proponen un enfoque para el modelado a bajo nivel del comportamiento del oponente de forma individual. En este

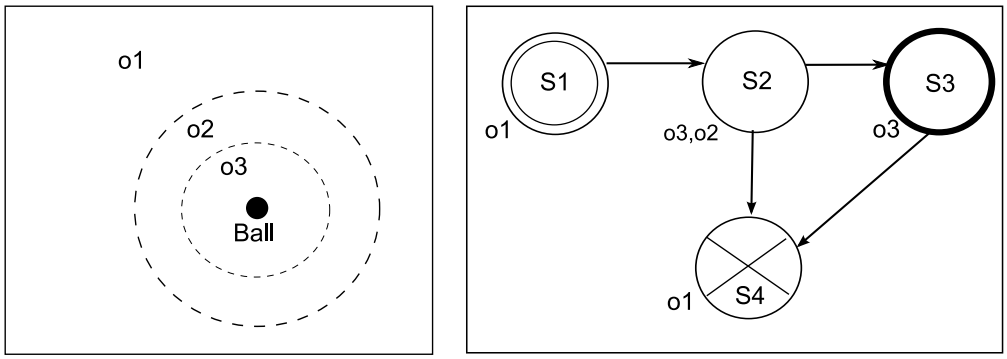
enfoque, es necesario como primer paso desarrollar un clasificador para inferir y etiquetar las acciones del oponente basándose en la observación. Este clasificador será entonces utilizado para etiquetar las acciones observadas por un agente. A partir de estas observaciones y utilizando técnicas de aprendizaje automático, se genera un modelo capaz de predecir las acciones del oponente. La arquitectura creada a partir de este enfoque se denomina OMBO (*Opponent Modeling Based on Observation*) y es utilizada y evaluada por los delanteros de un equipo para anticiparse a las acciones del portero.

Steffens [171] propone un enfoque de razonamiento basado en casos (*Case-Based Reasoning - CBR*) para el modelado del oponente que utiliza las ventajas del aprendizaje basado en instancias (*lazy learning*). En este caso, al aplicar *CBR* al modelado de agentes, la similitud de situaciones no puede calcularse de forma uniforme, sino que necesita adaptarse a la situación y rol de los agentes cuya acción se va a predecir. Por ejemplo, se considera que la posición de un defensa del equipo B será irrelevante si la clasificación que se quiere realizar es la acción del portero del equipo A. Sin embargo, será muy relevante si se quiere clasificar la acción de un delantero del equipo A. Por este motivo, algunos enfoques basados en *CBR* en el fútbol simulado resuelven este problema considerando que cada caso sólo afecta a aquellos jugadores cercanos a la pelota [4].

Además del fútbol simulado, la RoboCup ofrece otros dominios que utilizan agentes físicos y en los que también se debe reconocer el comportamiento del oponente para responder de forma inteligente en el entorno. Sin embargo, en estos dominios, la obtención de los eventos de los oponentes es un proceso más complejo. Tratando de abordar este problema, Vail y Veloso [181] proponen un método de selección de las características más relevantes del campo de fútbol en la competición de la *RoboCup Small Size* (tamaño pequeño) que permita crear de forma más eficiente el modelo de los oponentes.

En este mismo dominio, Han y Veloso [87] utilizan HMM para reconocer el comportamiento de agentes tanto software como físicos. En este caso, los estados de los HMM se corresponden con una descomposición abstracta de su comportamiento y las probabilidades de los estados intermedios son un indicador del comportamiento que se está produciendo. Estas probabilidades pueden utilizarse para anticiparse así a los estados futuros del robot. Las transiciones entre estados son probabilidades obtenidas mediante las observaciones del *mundo real*.

Un ejemplo sobre esta representación se muestra en la figura 2.1 en la que a partir de las observaciones del *mundo real* se construye un HMM que las representa. En este ejemplo, si un jugador se encuentre alejado de la pelota (*o1*), se puede considerar que éste está en el estado *S1*. Si dicho jugador se aproxima (*o2* o *o3*), éste pasará a un estado diferente, *S2*. Por último, desde este estado podrá ser poseedor de la pelota pasando así al estado *S3* o alejarse y transitar hacia un estado que hace que el comportamiento “Ir por el balón” no sea reconocido. En



Observaciones para el *comportamiento* "Ir a por el balón"

HMM que representa el *comportamiento* "Ir a por el balón"

Figura 2.1: Construcción del HMM a partir de las observaciones del *mundo real*

este trabajo, se utiliza un HMM por cada comportamiento que se desea representar. También es posible crear un HMM para el comportamiento de los robots en conjunto pero el modelo construido podría ser muy complejo.

2.2.4. Modelado de Usuario

El modelado de usuario es un área de investigación muy relacionada con el modelado de agentes que ha tenido un gran auge en las dos últimas décadas. Según Zukerman y Albrecht [198] modelar un usuario consiste en deducir información no observable sobre un usuario por medio de la información observable de dicho usuario. Por otra parte, Kay y McCreath [109] proponen la siguiente definición: *"Un modelo de usuario es una representación explícita de información sobre un usuario o un grupo de usuarios. Esta información es útil para mejorar la interacción entre el entorno y el usuario o grupo de usuarios. La representación debe permitir que el modelo de usuario pueda ser interpretado tanto por máquinas como por humanos."*

El modelado de usuario está muy relacionado con el campo de la interacción hombre-máquina que tiene como finalidad hacer más fácil la comunicación entre los usuarios y las máquinas. Muchas de estas técnicas que son actualmente empleadas para crear la mayoría de las interfaces de usuario, se originaron en las décadas de los 1960 y 1970 [139]. La interacción hombre-máquina es un campo de investigación multidisciplinar que está influenciado por una gran cantidad de áreas, como las ciencias de la computación (diseño de aplicaciones), la psicología (aplicación de teorías del proceso cognitivo y análisis empíricos del comportamiento del usuario), la antropología (interacciones entre tecnología, trabajo y organización), etcétera.

Actualmente, una de las líneas de investigación abiertas en este área se centra

en que no todos los comportamientos, intenciones y expectativas de los usuarios son iguales cuando interaccionan con un ordenador. Es decir, las interfaces son las mismas para los usuarios, pero no todos los usuarios poseen las mismas características.

En los primeros trabajos sobre modelado de usuario, se utilizaban bases de conocimiento realizadas a mano para hacer inferencias sobre observaciones de otros usuarios. En particular, estos sistemas usaban bibliotecas de planes creadas manualmente para inferir las intenciones o preferencias de los usuarios a partir de sus palabras [33]. Sin embargo, estas bases de conocimiento tienen principalmente dos problemas: su construcción es un proceso complicado y no exacto, y además no suelen ser adaptables o ampliables. Con la finalidad de resolver estos problemas, se han considerado principalmente dos enfoques, el aprendizaje automático y los modelos estadísticos de predicción.

- Si se considera el aprendizaje automático tal y como muestra Webb [186] para el modelado de usuarios, los modelos que se crean tratan de describir uno de los cuatro conceptos siguientes:

  1. El proceso cognitivo que hay detrás de las acciones del usuario.
  2. Las diferencias entre las aptitudes de un usuario experto y uno que no lo es.
  3. Las patrones de comportamiento de un usuario o sus preferencias.
  4. Las características de los usuarios.

  Las primeras investigaciones que utilizaban aprendizaje automático con este fin, se centraron principalmente en describir los dos primeros conceptos. Sin embargo, trabajos más recientes están enfocados al tercero de los enfoques, el comportamiento de un usuario y sus preferencias. Así, en [185] Webb presenta un paradigma para el modelado basado en características que se centra en la adquisición de patrones de comportamiento del usuario y en no intentar modelar el proceso cognitivo subyacente. De esta forma, si se pretende modelar un usuario en un sistema en el que dicho usuario debe realizar una serie de acciones repetidamente a partir de una serie de opciones predefinidas, el aprendizaje automático es una técnica muy útil para ello. Como ejemplo, se puede considerar el trabajo de Segal y Kephart [164] en el que se propone un clasificador de correos electrónicos que utiliza un procesador de textos para predecir las carpetas destino de los mensajes recibidos. En otro trabajo relacionado, Billsus y Pasan [26] establecen un sistema que determina qué nuevos artículos en web son adecuados para una persona utilizando para ello aprendizaje automático multi estrategia.

- Por otra parte, los modelos estadísticos de predicción son utilizados para predecir las acciones que realizará el usuario. Este enfoque abarca diferentes modelos, entre los que se encuentran; los modelos lineales [146], modelos basados en *TFIDF* (*Term Frequency Inverse Document Frequency* - Frecuencia de términos inversa a la frecuencia de documentos) [22], modelos de Markov [6] o redes bayesianas [98].

Por último, se debe considerar que el modelado de usuario es de gran utilidad en áreas en las que se requiere una rápida valoración del conocimiento del usuario, sin necesidad de que sea totalmente exacta. La importancia de añadir esta capacidad en los sistemas actuales se demuestra por la gran cantidad de áreas en los que ya se ha aplicado este tipo de modelado. A continuación, se detallan algunas de estas áreas.

- *Software educacional:*
  Los estudiantes se modelan para personalizar su proceso de aprendizaje con la finalidad de crear entornos que puedan adaptarse a las necesidades y conocimientos de cada estudiante. Sison y Shimura [168] examinan cómo las técnicas de aprendizaje automático pueden utilizarse para crear modelos de estudiantes de forma automática así como el conocimiento necesario para crear dichos modelos.

- *Minería de la Web:*
  La minería de la web (*Web Mining*) es el proceso global de extracción de información previamente desconocida a partir de datos de la Web utilizando para ello técnicas de minería de datos [64]. Taxonómicamente, la minería web se divide en tres tipos [115]:

  - *Minería web de contenido:* Analiza el contenido de los documentos Web. Dicho contenido puede ser texto plano, código HTML u otras estructuras más complejas de información como audio o vídeo.
  - *Minería web de estructura:* Extrae información del análisis de la estructura de hiperenlaces entre sitios web.
  - *Minería web de uso:* Analiza el comportamiento de los usuarios en su interacción con la Web. Para ello, utiliza ficheros de trazas contenidos en servidores Web y archivos *cookie* en los ordenadores de los usuarios.

  Dentro de la Minería web de uso, el descubrir y extraer los patrones de navegación interesantes es una de las tareas más demandadas en los servicios de personalización. Con esta finalidad, se han propuestos varios esquemas.

Spiliopoulou y Faulstich [170] presentan *Web Utilization Miner (WUM)*, un sistema de minería de datos para detectar los patrones de navegación interesantes en las páginas web. *WUM* procesa todos los ficheros generados por la web y utiliza un lenguaje (*MINT*) capaz de detectar dichos patrones con la ayuda de un experto. Wu et al. [193] presentan un mecanismo independiente del portal basado en HMM capaz de personalizar los intereses de los usuarios a partir de sus patrones de navegación.

En este mismo entorno, la construcción de un motor de búsqueda debe considerar dos aspectos importantes: ¿cómo localizar los documentos relevantes en la red? y ¿cómo filtrar los no relevantes?. Considerando estas cuestiones, Godoy y Amandi [78] presentan una técnica para generar perfiles de usuario capaces de ser entendidos por los humanos que codifican sus intereses mediante la observación de su comportamiento en la web. La técnica propuesta es construida basándose en el algoritmo conocido como *agrupamiento conceptual de documentos web*.

- *Sistemas de recomendación web:*
  Los sistemas de recomendación suelen considerarse como un tipo concreto de técnica para filtrar información y poder así presentar al usuario la más relevante. Macedo et al. [134] proponen un sistema (*WebMemex*) que captura información de la red (como direcciones IP, identificadores de usuario y direcciones web visitadas) y es capaz de recomendar páginas webs relevantes para el usuario.

2.3. Minería de secuencias de datos

Todas las acciones que realiza un agente se producen de forma secuencial, por lo que en muchos casos, dicha secuencialidad es muy relevante para la construcción de los modelos. Por ejemplo, cuando un usuario interacciona con una máquina se establece un determinado *diálogo*, el cuál es intrínsecamente secuencial. Si se quisiera modelar el comportamiento de este usuario en función de su interacción con la máquina, se debe tener en cuenta la secuencialidad de dicho *diálogo*.

En particular, la minería de secuencias de datos puede considerarse como un subcampo de la minería de datos que opera sobre datos estructurados [59]. Datos estructurados son aquellos en los que cada dato es una composición explícitamente estructurada de un conjunto de elementos. Otros tipos de datos estructurados son aquellos que se distribuyen en una estructura de árbol, en un grafo, en series temporales o como texto. Estas estructuras añaden cierta información a los datos que los representan: órdenes temporales, estructuras jerárquicas y de red, etcétera.

En el caso de la minería de secuencias de datos, la información que aporta su estructura es el orden en el que dichos datos están representados. Este tipo de minería de datos es un componente esencial en diversos dominios de aplicación; como por ejemplo, la planificación, la robótica, el procesamiento del lenguaje natural, la predicción de series temporales, el control adaptativo, el tratamiento de secuencias de ADN, etcétera.

En este apartado se presenta de forma detallada la definición de secuencia considerada en esta tesis y porqué se propone la minería de secuencias de datos para el modelado de agentes. También se revisan los trabajos realizados sobre aprendizaje automático en secuencias de elementos.

2.3.1. Definición y ámbito

De acuerdo con la definición dada por el diccionario de la Real Academia Española, una secuencia es *una serie o sucesión de cosas que guardan entre sí cierta relación*. El diccionario Merriam-Webster define secuencia como *un conjunto de elementos ordenados y que pueden ser etiquetados con un número entero positivo*.

Para esta tesis, cualquiera de las dos definiciones anteriores son válidas y así, una secuencia de *n* elementos se define como *Secuencia* = $\{e_1, e_2, \ldots, e_n\}$ que determina una lista de elementos de longitud *n* estrictamente ordenada. El orden de los elementos denota que se trata de elementos contiguos.

En determinadas aplicaciones, las observaciones o ejemplos son inherentemente secuenciales. Es decir, la observación está formada de múltiples componentes que están relacionados entre sí y estrictamente ordenados en una dimensión, que normalmente suele ser el tiempo o el espacio. Por ejemplo, si se considera la dimensión del espacio, las secuencias de ADN están formadas por una secuencia de diferentes tipos de nucleótidos por lo que éstas se pueden representar como una secuencia de caracteres A, C, G y T. El orden espacial que adoptan estas bases determina la estructura del ADN. Considerando la dimensión del tiempo, se pueden considerar los comandos introducidos por un usuario en una interfaz de comandos donde el orden de dichos comandos es esencial.

En el modelado de agentes, la minería de secuencias es relevante porque un agente considera y almacena los conceptos percibidos de una forma relacional o contextual. Por ejemplo, considérese la interacción de un usuario con su ordenador. Si el ordenador del usuario se reinicia varias veces, el usuario podría considerar que su ordenador está estropeado. Sin embargo, si el ordenador siempre se reinicia después de abrir una determinada aplicación, el usuario consideraría que dicha aplicación no funciona correctamente.

Por otra parte, relacionado con el modelado de agentes, y más concretamente con el modelado de usuarios, la secuencialidad de las acciones es el aspecto más importante que se debe considerar en el aprendizaje de habilidades en los

humanos [189]. Anderson [10] destaca que las secuencias de elementos son fundamentales para el ser humano en la resolución de problemas.

Existen tareas muy diversas en la minería de secuencias de datos en función de la finalidad que se persigue. A continuación se detallan las más relevantes:

- *Descubrimiento de subsecuencias relevantes* (patrones):
  Dada una secuencia, la finalidad de esta tarea consiste en extraer aquellas subsecuencias que son más representativas y que por lo tanto pueden representar un determinado patrón en dicha secuencia. Dillon et al. [179] presentan un algoritmo, denominado SEQUEST, que realiza este tipo de tarea y que es de especial relevancia cuando la cantidad de subsecuencias que se desean tratar es muy elevada.

- *Predicción de secuencias*:
  En esta tarea se pretende predecir elementos de una secuencia basándose en sus precedentes. Laird et al. [120] presentan un algoritmo, basado en métodos de compresión de texto, para predecir secuencias.

- *Clasificación de secuencias*:
  Esta tarea se basa en la construcción de un clasificador a partir de un conjunto de secuencias de entrenamiento etiquetadas. Bicego et al [25] presentan un método de clasificación de secuencias basado en Modelos de Markov y que se aplica en el reconocimiento de formas y de caras.

- *Agrupación de secuencias*:
  Dado un conjunto de secuencias no etiquetadas, esta tarea crea subconjuntos de forma que aquellas secuencias *similares* se encuentren en el mismo grupo. Para realizar esta división, es necesario definir el concepto *similar* empleado para evaluar la calidad de la agrupación. Yang et al. [195] proponen un modelo para agrupar secuencias en función de la relevancia de ciertos valores estadísticos.

2.3.2. Aprendizaje automático en secuencias de elementos.

Como se ha comentado anteriormente, las secuencias biológicas permiten entender las estructuras y funciones de moléculas para poder así, entre otras cosas, diagnosticar y tratar enfermedades. En concreto, una secuencia de ADN está formada por una secuencia de nucleótidos - Adenina (A), Guanina (G), Timina (T) y Citosina (C) - y la disposición secuencial de estos cuatro elementos a lo largo de la cadena es la que codifica la información genética. Ma et al. [133] presentan

una técnica novedosa para la clasificación de bio-secuencias. Dada una secuencia *S* de ADN no etiquetada, esta técnica trata de determinar si la secuencia *S* forma parte del ADN de la bacteria *E. Coli* (bacteria necesaria para el funcionamiento correcto del proceso digestivo). Este problema puede considerarse como un problema de clasificación [183] en el que se utilizan como entrada secuencias, en este caso de ADN. Loewenstern et al. [131] en sus investigaciones sobre este tema, describen *CBI* (*Compression-Based Induction*), un método de clasificación de secuencias basado en la utilización de técnicas de compresión de texto en secuencias de ADN. En este mismo dominio, Asai et al. [16] introducen un método para analizar las secuencias de aminoácidos de las proteínas utilizando un *Modelo Oculto de Markov* (HMM) de 3 estados. En un trabajo relacionado, Baldi et al. [21] describen las implicaciones estructurales de un patrón periódico encontrado en nucleosomas utilizando un HMM de 10 estados.

Por otra parte, un problema muy diferente como es la detección de intrusos en un ordenador, se puede tratar considerando las diferentes secuencias de comandos que se reciben como entrada un sistema. Coull et al. [52] proponen un algoritmo que utiliza el alineamiento de secuencias de pares relacionados para detectar similitudes entre las secuencias de comandos. Este algoritmo produce una medida efectiva para distinguir entre usuarios legítimos e intrusos.

Schoulau et al. [163] contrastan una serie de enfoques estadísticos para detectar intrusos en un sistema informático basándose únicamente en las secuencias de comandos tecleadas. Sus resultados muestran que este tipo de métodos estadísticos son muy eficaces en este entorno. Además, Lane y Brodley [121] proponen un enfoque para examinar secuencias de acciones de usuarios, en este caso comandos UNIX, y clasificar su comportamiento en normal o anómalo. En este enfoque se exploran diversas funciones que comparan una secuencia que representa un comportamiento observado con perfiles de usuario ya definidos. Además, se discuten las dificultades que conlleva la clasificación de datos generados por humanos.

Basándose en sistemas de aprendizaje automático, Weiss y Hirsh [187] proponen un algoritmo genético que predice eventos poco frecuentes. Este algoritmo identifica patrones secuenciales y se aplica en la predicción de caídas de equipos de telecomunicaciones a partir de sus mensajes de alarma.

En el tratamiento de secuencias también se han utilizado una serie de técnicas basadas en redes de neuronas artificiales (RNA). Frasconi et al. [69] y Giles et al. [77] utilizan en diferentes trabajos redes recurrentes con un método de entrenamiento basado en retropropagación para predicción de secuencias.

La importancia de los datos secuenciales en el modelado de agentes, ha inspirado otras investigaciones. Kaminka et al. [104] enfocan su trabajo en el aprendizaje autónomo no supervisado de secuencias de comportamiento de agentes a partir de observaciones de los propios agentes. Las técnicas utilizadas realizan la

conversión de observaciones de un estado del mundo complejo, continuo y muy variable a una serie de secuencias que representan comportamientos atómicos. Obtenidas las secuencias, éstas se analizan para encontrar las subsecuencias que caracterizan el comportamiento de cada agente. En este caso, el dominio de evaluación es el ofrecido por el simulador de fútbol de la *RoboCup*.

Horman y Kaminka [89] afrontan el problema de aprendizaje no supervisado en secuencias de elementos. Para ello, comparan empíricamente diferentes enfoques en dos tareas determinadas: la detección de *ruido* en textos y la detección de patrones que definan los comportamientos de un usuario en una interfaz de comandos.

2.4. Sistemas Auto-Adaptativos (*Evolving Systems*)

En este apartado, se resumen los fundamentos, metodología, diseño, aspectos de implementación y aplicaciones del paradigma de los Sistemas Auto-Adaptativos.

2.4.1. Motivación

En los últimos años, las necesidades en el tratamiento de información están sufriendo grandes cambios. Muchos sistemas actuales de toma de datos generan gran cantidad de información de forma continua. Por ejemplo, aquellos sistemas que recogen datos de los procesos industriales, los mercados de consumo, Internet, etcétera. Sin embargo, en muchos entornos no es posible o es poco viable almacenar datos para su posterior análisis por lotes. De esta forma, los retos a los que hacer frente en el procesado de información, y en particular en su clasificación, están relacionados con:

1. Necesidad de tratar y analizar grandes cantidades de datos.

2. Procesar flujos de datos *on-line* y/o en tiempo real [58].

Uno de los paradigmas que trata de crear sistemas que aborden estos dos aspectos, son los denominados (en inglés) *Evolving Systems* [15]. A pesar de que las primeras investigaciones en estos sistemas datan del año 2000, no existe una traducción al español utilizada en el ámbito científico. En esta tesis se ha optado por utilizar el término Sistemas Auto-Adaptativos (SAA).

2.4.2. Definición

Los SAA son sistemas que pueden desarrollarse y aprender por sí mismos y en los que tanto sus parámetros como su estructura se adaptan al entorno conforme

éste varía. Debido a sus características, estos sistemas son capaces de recibir, tratar y analizar gran cantidad de datos en tiempo real.

Normalmente, las técnicas *adaptativas* convencionales son adecuadas para representar sistemas que sufren pequeños cambios en su estructura. Sin embargo, para el manejo de sistemas complejos con múltiples modos de operación o cambios drásticos en sus características, estas técnicas convencionales suelen necesitar mucho tiempo para aprender los nuevos parámetros del modelo. Por su parte, los SAA son capaces de evolucionar (expandir o reducir) su estructura, su funcionalidad y su representación interna del conocimiento para adaptarse así a los cambios en el entorno [15]. La base de estos sistemas son los sistemas borrosos basados en reglas.

Los SAA pretenden generar respuestas con los datos recibidos en tiempo real. Dada la gran cantidad de datos que se pueden recibir en tiempo real, estos sistemas desechan todos aquellos datos que no consideran relevantes. Así, no necesitan almacenar todos los datos que se reciben, ya que determinadas variables almacenan cierta información que permite omitir gran cantidad de datos sin disminuir la calidad del sistema. Esta característica hace que no sea posible deshacer cambios en el sistema ni obtener situaciones anteriores a la actual; sin embargo, permite procesar los datos a gran velocidad.

Otra característica importante de los SAA es su posible interpretación; es decir, los modelos que obtienen puede ser interpretados por los humanos.

2.4.3. Diferencia con otros sistemas existentes

Existen otros sistemas que cumplen determinados aspectos considerados por los SAA; sin embargo, ninguno de ellos los cumple en su totalidad. A continuación se detallan algunos de estos sistemas.

Sistemas Auto-Adaptativos frente a Sistemas Evolutivos

En la actualidad, los algoritmos evolutivos están bien definidos y son capaces de buscar una solución a un problema basándose en técnicas computacionales relacionadas con la evolución de los individuos que forman una población [88].

Sin embargo, los SAA hacen referencia a la capacidad que tiene cada individuo de desarrollarse por sí mismo, algo que suele ser conocido en los humanos como desarrollo mental autónomo. En los SAA se crea nuevo conocimiento a partir del aprendizaje de las experiencias por lo que son comparables con el proceso de aprendizaje gradual de los seres humanos a lo largo del tiempo. De esta forma, al igual que el proceso de aprendizaje de los humanos, estos sistemas pueden inicializarse con determinado conocimiento o aprender sin una base inicial.

Sistemas Auto-Adaptativos frente a Sistemas Incrementales

En minería de datos, el problema de analizar y clasificar un flujo continuo de datos se resuelve con diversos enfoques. Uno de estos enfoques se basa en clasificadores incrementales implementados con técnicas muy diferentes:

- Reglas de decisión [67].
- Métodos basados en similitudes entre ejemplos: redes de cuantización vectorial incrementales, iLVQ [194] y teoría de resonancia adaptativa, ART [39]
- Métodos probabilísticos: clasificadores incrementales bayesianos [41] y análisis incremental de componentes principales, iPCA [147].

Todos los algoritmos anteriores mantienen una estructura fija que debe adaptarse a todo tipo de cambios en los datos. Sin embargo, si estos cambios son muy bruscos, la estructura predefinida también debería cambiar bruscamente. Los SAA proponen una estructura capaz de modificarse en función de los datos recibidos. Kasabov y Song [107] proponen una aproximación a este concepto mediante una red de neuronas con una estructura auto-adaptativa, pero sólo es aplicable a la predicción de series temporales.

2.4.4. SAA como Sistemas Borrosos Basados en Reglas

Los SAA están basados en sistemas borrosos por lo que están muy relacionados con la forma en que los humanos razonan [197]. Estos sistemas permiten formalizar matemáticamente la incertidumbre, la información subjetiva, las preferencias, la experiencia e incluso la intuición, valores que de otra forma serían muy difíciles o imposibles de describir. Así, los humanos se rigen por reglas del tipo: «*Si llueve, cogeré un paraguas*», o «*Si la distancia es corta, iré andando*»; donde *llueve* y *corta* son términos no definidos de forma concisa que adquieren valores *borrosos* definidos por conjuntos borrosos. Este proceso de razonamiento hace que un humano cree, adapte, elimine o reemplace reglas. Por ejemplo, después de coger un paraguas un día de lluvia con mucho viento, la regla aprendida «*Si llueve, cogeré un paraguas*» podría ser modificada por «*Si llueve y no hace viento, cogeré un paraguas*».

El marco estructural utilizado por los SAA son los sistemas borrosos basados en reglas del tipo Takagi-Sugeno, TS [175]. Estos sistemas pueden ser descrito por reglas de la forma *si-entonces* con una estructura de antecedente y consecuente definida siguiente modo:

$$Regla_i = SI\ (x_1\ es\ similar\ a\ x_{i1})\ Y\ (x_2\ es\ similar\ a\ x_{i2})\ Y\ ...$$

$... Y\ (x_n\ es\ similar\ a\ x_{in})\ ENTONCES(y_i = a_{i0} + a_{i_1x_1} + ... + a_{in}x_n)$

donde:

- $Regla_i$ es una de las *R* reglas borrosas (*i=1,2,...,R*) almacenadas en la base de reglas.
- x_j (*j=1,2,...,n*) representan las variables de entrada.
- y_i denota la salida de la regla borrosa *i*.
- x_{ij} representa el prototipo (concepto que será explicado más adelante) de la regla borrosa *i*.
- a_{ij} es el parámetro *j* de la regla borrosa *i*.

Tal y como se muestra en la Figura 2.2, se observa que los sistemas TS borrosos pueden ser representados como una red de neuronas artificiales y que por lo tanto son sistemas neuro-borrosos [11].

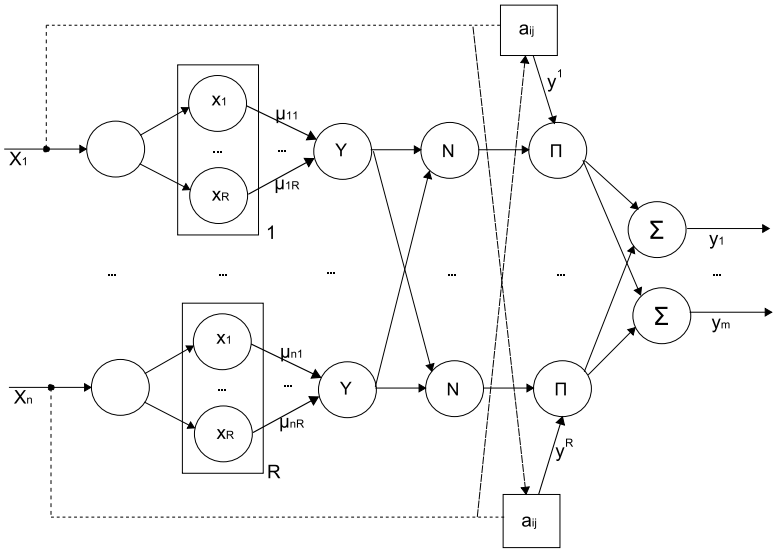


Figura 2.2: Sistema TS Borroso representado como un sistema neuro-borroso.

Los elementos de los sistemas borrosos TS son:

1. Antecendentes, Parte *SI*:
   - Los distintos conjuntos borrosos están conectados con operadores lógicos (Y y O) y podrían representar variables lingüísticas (por ejemplo: X_{i1} es alto, X_{i2} es pequeño...).

- La función de pertenencia de estos conjuntos borrosos (representan el grado de pertenencia de un particular valor en un conjunto borroso específico, μ_{ij}).
- Parámetros de la función de pertenencia.
- Número de variables de entrada.

2. Consecuentes, Parte *ENTONCES*:
   - Parámetros de la parte consecuente (normalmente lineal), por ejemplo a_{ij}.
   - Número de salidas, *m*.
3. Número de reglas borrosas, *R*.

De esta forma, los SAA se basan en que tanto las reglas como sus variables puedan evolucionar, cambiando gradualmente. Así, en función de los datos recibidos se aumentará o disminuirá tanto el conjunto de reglas creados por el sistema como el conjunto de variables que forman cada una de dichas reglas.

2.4.5. Otros enfoques relacionados

Los problemas más relevantes que pueden resolverse utilizando SAA son el agrupamiento, la predicción de series temporales y la clasificación. Dado que en esta tesis doctoral los SAA se utilizan para problemas de agrupamiento y clasificación, en este apartado se detallan tres algoritmos con ciertas semejanzas a los clasificadores basados en SAA. Sin embargo, éstos no son capaces de tratar y analizar gran cantidad de datos de forma incremental y poder modificar tanto su estructura como sus parámetros en función de los datos.

- **Razonamiento Basado en Casos**

  El *razonamiento basado en casos* (*Case-Based Reasoning, CBR*) es una técnica de resolución de problemas que se fundamenta en la premisa de que problemas similares tienen una solución similar [1]. Con esta técnica se intentan resolver nuevos problemas mediante la adaptación de soluciones que fueron utilizadas con anterioridad en la resolución de problemas similares [50, 54]. En los clasificadores basados en casos, la clase a la que pertenecerá un caso no etiquetado se determina examinando la clase de los *k* casos más cercanos en la base de casos.

  Uno de los algoritmos más conocidos para solucionar este tipo de problemas es el *vecino más cercano* (*Nearest Neighbor algorithm*) [53]. Este algoritmo compara un nuevo caso con todos los observados previamente y le asigna

la clase del caso más similar. Sin embargo, este tipo de clasificación debe considerar ciertos problemas:

- Los casos deben representarse de forma que la distancia entre ambos mida de forma adecuada su similitud.
- El coste de la clasificación puede ser muy elevado porque cada caso debe compararse con una gran cantidad de casos almacenados.
- La *base de casos* debe almacenar todos los datos observados por lo que su tamaño puede ser considerable.
- El ruido en la *base de casos* puede reducir la precisión de la clasificación.

La principal diferencia entre este tipo de algoritmos y los SAA está en que estos últimos no necesitan almacenar todos los casos observados. Por este motivo, los SAA no pueden deshacer cambios, pero manejan y procesan grandes cantidades de datos en tiempo real de forma eficiente.

- **Redes de Cuantización Vectorial (LVQ)**

  Una red de cuantización vectorial (LVQ) es un algoritmo de clasificación supervisada basado en el cálculo de prototipos a partir de un conjunto de entrenamiento y que realizan la clasificación por vecindad. Este algoritmo está basado en el algoritmo de mapas auto-organizados (*Self-Organizing Map, SOM*) desarrollado por Kohonen [113]. Inicialmente es necesaria una fase de aprendizaje utilizando ejemplos de entrenamiento. Como en el modelo de *Kohonen*, los prototipos se desplazan en función de los ejemplos recibidos [76]. La solución dependerá de la colocación final de los prototipos y de cómo se mida la distancia entre ellos.

  El algoritmo LVQ tiene como ventajas su rapidez en el entrenamiento, su capacidad para utilizar grandes cantidades de datos y su posibilidad de manejar vectores que no contengan todos los valores. Una de las muchas aplicaciones en la que se utilizan estos sistemas es la clasificación de documentos de texto [180]. Sin embargo, el modelo creado utilizando LVQ tiene una serie de desventajas:

  - No puede manejar datos incrementales.
  - La precisión depende de la distribución de las clases en el conjunto de entrenamiento, por lo que si no se tiene información previa sobre los datos, este valor será arbitrario.
  - El resultado depende de sus valores iniciales y de los parámetros de aprendizaje utilizados.

Algoritmo utilizado por LVQ

Entradas:
Conjunto de datos de ejemplo de entrenamiento clasificados
Salida:
Conjunto de prototipos actualizados que definen los ejemplos de entrada

Procedimiento:

```
1. Decidir el número de prototipos e inicializarlos
aleatoriamente.
2. Introducir un nuevo ejemplo y calcular su distancia a todos los
prototipos.
3. Elegir el prototipo más cercano y etiquetar como el ganador.
4. Si el nuevo ejemplo y el ganador pertenecen a la misma clase,
desplazar el prototipo ganador hacia el patrón.
5. Si el nuevo ejemplo y el ganador no pertenecen a la misma
clase, alejar el prototipo ganador del ejemplo.
6. Si los prototipos aún se desplazan, o el valor alfa es mayor
que cero, ir a 2.
```

Figura 2.3: Algoritmo utilizado por LVQ

Con la finalidad de solventar la primera de estas desventajas, Xu et al. [194] proponen iLVQ, un algoritmo LVQ incremental para tareas de clasificación supervisada capaz de manejar conjuntos de datos incrementales. A pesar de que iLVQ es un método incremental, a diferencia de los SAA, su estructura es fija.

- ***Teoría de Resonancia Adaptativa (ART)***

  La Teoría de Resonancia Adaptativa *(Adaptive Resonance Theory, ART)* es una teoría desarrollada por Carpenter y Grossberg [38] basada en el mecanismo que utiliza el cerebro para procesar información.

  Los sistemas *ART* afrontan la problemática entre plasticidad[2] y estabilidad[3]. Un modelo de RNA capaz de resolver uno de estos aspectos es sencillo, pero obtener un sistema capaz de dar respuesta a ambos, no lo es. ART resuelve

[2]Capacidad para aprender nuevos patrones
[3]Capacidad para retener los patrones aprendidos

este problema al hacer posible que el aprendizaje ocurra solamente en un estado de resonancia. Es decir, cuando a la red se le presenta un patrón de entrada, éste se hace resonar con los prototipos de las categorías conocidas por la red, si el patrón entra en resonancia con alguna clase, entonces se asocia a ésta y el centro de la clase es desplazado ligeramente. En caso contrario, y si la red tuviera una capa de salida dinámica, se creará una nueva clase para dicho patrón sin afectar a las otras clases ya existentes.

Para que este método pueda tratar datos de forma incremental, Carpenter et al. [39] proponen el algoritmo *fuzzy ARTMAP* que se basa en lógica difusa y en RNA basadas en ART. Sin embargo, la estructura de *fuzzy ARTMAP*, a diferencia de la propuesta por los SAA, no es capaz de modificarse en función de los datos obtenidos a lo largo del tiempo.

2.5. Conclusiones

Tanto en dominios competitivos como colaborativos, el poder adquirir conocimiento de otros agentes mediante su observación es una capacidad que puede ser de gran ayuda en ámbitos muy diferentes. En este capítulo se han presentado las investigaciones relacionadas con el modelado de agentes que engloban todo tipo de técnicas aplicadas en distintos dominios.

En la mayoría de las investigaciones detalladas, el modelado que proponen es dependiente del dominio. Muchos de estos métodos de modelados utilizan determinadas características propias de un dominio en concreto. En esta tesis doctoral, se proponen distintos enfoques de modelado de agentes lo más generales posibles con la finalidad de que puedan utilizarse en dominios muy diversos.

Otros de los enfoques propuestos para el modelado y reconocimiento de diferentes comportamientos de agente utilizan autómatas finitos deterministas. A diferencia de estos enfoques, esta tesis doctoral se enfoca en dominios en los que el comportamiento de un agente observado es no determinista y lo suficientemente complejo como para no poder ser modelado por un autómata finito de un tamaño aceptable.

En determinadas investigaciones se trata el tema de reconocimiento de acciones utilizando enfoques estadísticos. Sin embargo, dichos trabajos se centran en aprendizaje no supervisado, sin capacidad para clasificar un comportamiento en una determinada clase.

En este capítulo también se han expuesto una serie de estudios que presentan técnicas para inferir diferentes tipos de conocimiento a partir de secuencias de elementos debido a que el comportamiento de todo agente está completamente asociado con la secuencialidad de las acciones que éste realiza. Incluso en el

aprendizaje de habilidades en humanos, la secuencialidad de elementos es un factor fundamental, de forma que tanto en el modelado de usuario como en el de agentes, se debe considerar esta secuencialidad. Las investigaciones referentes al aprendizaje automático en secuencias de elementos están muy relacionadas con esta tesis doctoral porque la base de los enfoques propuestos consiste en considerar el comportamiento de agentes como la secuencia de elementos que lo representa.

Existen ciertos enfoques en el modelado de agentes que utilizan HMM y por lo tanto consideran que la probabilidad de pasar de un estado a otro es independiente de los estados previos. El enfoque propuesto en esta tesis doctoral, tiene en cuenta secuencias cortas de eventos que incorporan cierto contexto histórico en el modelo del agente.

Por último se ha presentado un tipo de sistemas denominados Auto-Adaptativos que son capaces de modificar tanto su estructura como sus parámetros en función de los datos recibidos. Estos sistemas comenzaron a desarrollarse en el año 2000 y son útiles en el modelado de agentes porque el comportamiento de todo agente puede cambiar con el tiempo y el sistema que lo modela debe ser capaz de adaptarse a dichos cambios.

Capítulo 3

Modelado de Agentes Mediante Comparación de Secuencias de Eventos: M-COMP

En este capítulo se detalla el primero de los enfoques propuestos en esta tesis doctoral, denominado M-COMP. Este enfoque tiene como finalidad modelar, detectar y comparar comportamientos de un conjunto de agentes en determinados entornos. A pesar de que se propone un enfoque general, su implementación está totalmente orientada a la resolución de la problemática propuesta en la competición *RoboCup Coach 2006* [184] y cumple de forma estricta las reglas de dicha competición.

Uno de los objetivos del enfoque planteado, y por lo tanto de esta competición, consiste en modelar el comportamiento de un conjunto de agentes a partir de la observación de su comportamiento. Sin embargo, en este caso, dicho modelo de comportamiento se obtiene a partir de la observación de dos comportamientos diferentes y su posterior comparación.

En la competición en la que se centra este enfoque, *RoboCup Coach 2006*, el conjunto de agentes está representado por un equipo de fútbol simulado. A pesar de ello, M-COMP se presenta como un enfoque general y se explica como tal.

La Figura 3.1 muestra un esquema general del funcionamiento de M-COMP:

- La primera parte consiste en la obtención de secuencias de eventos a partir de la observación del comportamiento de un conjunto de agentes. Esta conversión es realizada en el MCSE (*Módulo de Creación de Secuencias de Eventos*) y es el único proceso dependiente del dominio. Este módulo se detalla en el apartado 3.1

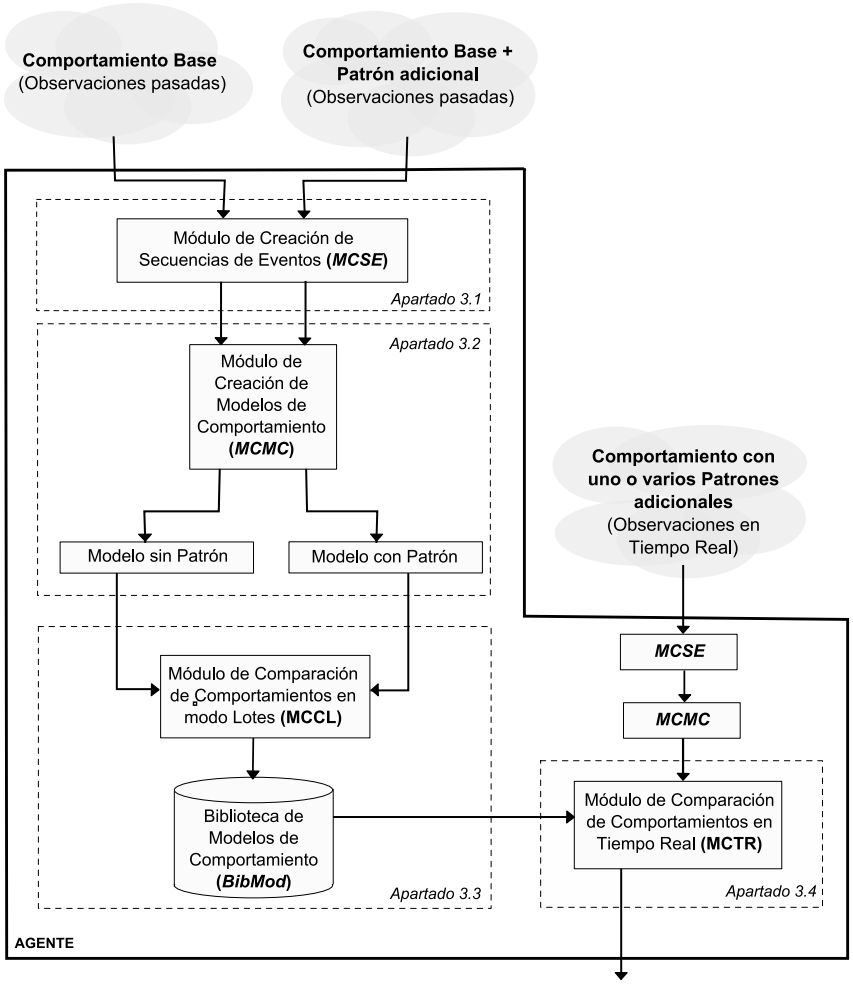


Figura 3.1: Estructura general de M-COMP - Modelado de Agentes Mediante Comparación de Secuenciación de Eventos.

- La segunda parte consiste en la construcción del modelo de comportamiento de un conjunto de agentes a partir de su observación. Este proceso se realiza en el MCMC (*Módulo de Creación de Modelos de Comportamiento*) y se explica en el apartado 3.2. El proceso se aplica por lotes y su principal característica es que se basa en el análisis de las secuencias de los eventos que realiza el conjunto de agentes cuyo comportamiento se quiere modelar. En este caso, y considerando las observaciones recibidas, los modelos construidos serán dos: *Modelo sin Patrón* y *Modelo con Patrón*.

- Una vez obtenidos el modelo del *comportamiento base* (*Modelo sin Patrón*) y el modelo del *comportamiento base + patrón* (*Modelo con Patrón*), éstos se deben comparar para detectar sus diferencias y determinar así el patrón

de comportamiento que se ha añadido al *comportamiento base*. El resultado de esta comparación es el modelo de comportamiento que se almacena en la biblioteca denominada BIBMOD. El módulo que implementa este proceso se denomina MCCL (*Módulo de Comparación de Comportamientos en modo Lotes*) y se explica de forma detallada en el apartado 3.3.

- Por último, se observa un conjunto de agentes en tiempo real y considerando los modelos de comportamiento almacenados en BIBMOD, se detecta qué comportamientos está siguiendo dicho conjunto de agentes. Como esta detección se realiza en tiempo real, el módulo se denomina MCTR (Módulo de Comparación de comportamientos en Tiempo Real) y su funcionamiento se presenta en el apartado 3.4.

3.1. Creación de Secuencias de Eventos - MCSE

Este proceso recibe como entrada las observaciones de las acciones realizadas por el conjunto de agentes que se desea modelar. De acuerdo con las características de los agentes y el entorno, se deben extraer aquellos datos que permitan obtener una secuencia de los eventos llevados a cabo por dichos agentes. En este caso, un *evento* es una observación que ocurre en un determinado lugar durante un determinado intervalo de tiempo y que define una acción específica de uno o varios agentes. Los datos que deben extraerse del entorno y los eventos que se obtienen son dependientes del dominio.

Como ya se ha mencionado, una secuencia de eventos se define como: *Secuencia* = $A_1A_2...A_n$ donde se muestran *n* eventos contiguos y ordenados. La duración de cada uno de los eventos no es considerada en esta propuesta.

3.2. Creación de Modelos de Comportamiento - MCMC

Para modelar un determinado comportamiento a partir de la secuencia de eventos que lo representa, se puede considerar que aquellos eventos que más se repiten son los más relevantes para describir dicho comportamiento. Esta idea es muy utilizada en áreas como recuperación de información o minería de textos en las que medidas estadísticas como *tf-idf (term frequency-inverse document frequency)* son aplicadas para evaluar la importancia de una palabra en un documento o conjunto de documentos. Así, este valor se mide en función de las veces que aparece una palabra en un documento y se normaliza a partir de la longitud de dicho documento.

Sin embargo, en ciertos entornos las dependencias temporales entre los eventos que conforman un comportamiento también resultan muy significativas ya que todo evento depende de los anteriores y está relacionado con los posteriores. Por ejemplo, en una interfaz humano-ordenador el orden de los comandos enviados por el usuario es relevante para el resultado final de dichos comandos. Considérese la diferencia entre las secuencia de comandos UNIX: «`rm a.txt; mv b.txt a.txt`» y la secuencia: «`mv b.txt a.txt; rm a.txt`»

Teniendo en cuenta estos dos aspectos, el modelo de comportamiento que se propone se basa en una estructura que identifica de forma sencilla aquellos eventos más repetidos pero también su dependencia con eventos anteriores y posteriores. Esta estructura se denomina *trie* [70].

Un *trie* [1] es una estructura de árbol en la que los datos se almacenan de forma ordenada. Este tipo de árboles de búsqueda es similar al usado por las tablas de página en los sistemas de memoria virtual y, a diferencia de los árboles binarios de búsqueda [112], los nodos de un trie no almacenan la clave asociada a cada nodo porque ésta viene determinada por su posición en la estructura. Su principal aplicación es la representación de diccionarios grandes de palabras ya que puede disminuir la memoria necesaria para su almacenamiento y agilizar las operaciones que se realizan. En esta estructura, todos los descendientes de un nodo tienen un prefijo común asociado con dicho nodo, y la raíz de la estructura representa la cadena vacía. Un ejemplo de esta estructura se muestra en la Figura 3.2 en la que cada nodo representa una letra determinada pudiéndose almacenar así de forma eficiente palabras como «`da`», «`dos`», «`don`», «`doy`», «`a`» y «`mis`». La búsqueda de una palabra en este tipo de estructuras es muy rápida.

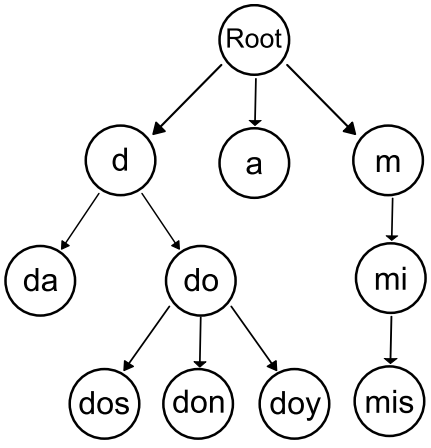


Figura 3.2: Ejemplo de *trie* utilizado para almacenar las palabras: «`da`», «`dos`», «`don`», «`doy`», «`a`» y «`mis`».

[1] Abreviación de *retrieval* (recuperación). También se conoce como *prefix tree*.

Esta misma estructura también se utiliza para tareas muy diferentes al ejemplo anterior debido a que también permite almacenar el número de veces que se ha insertado una determinada palabra. Por ejemplo, Kaminka et al. [104] utilizan un *trie* para detectar ciertos patrones en las acciones de un equipo de fútbol, mientras que Huang et al. [91] lo utilizan para crear patrones frecuentes en entornos que están en constante cambio.

En el MCMC se realiza un proceso que utiliza la estructura *trie* para la construcción de un modelo de comportamiento. Este proceso se divide en tres partes:

1. Segmentación de la secuencia de eventos.
2. Almacenamiento de las subsecuencias en el *trie*.
3. Cálculo de la relevancia de las subsecuencias.

Supóngase que observando un conjunto de agentes y su entorno se obtiene la siguiente secuencia de eventos: $\{A \rightarrow B \rightarrow A \rightarrow B \rightarrow C\}$, en la que cada letra mayúscula representa un evento diferente y la posición del evento en dicha secuencia representa el orden en el que éstos se han ejecutado.

1. Segmentación de la secuencia de eventos

La secuencia de eventos inicial se debe segmentar en varias subsecuencias con la finalidad de poder insertarse en un *trie* de manera separada. Esta segmentación se puede realizar utilizando alguna característica propia del dominio que permita obtener subsecuencias con un número reducido de eventos.

Si el entorno no proporciona una característica adecuada para esta división, la secuencia se puede segmentar en todas las posibles subsecuencias de eventos ordenados de la misma longitud. De esta forma, la secuencia:

A=$A_1A_2...A_n$ (n = número de eventos que contiene la secuencia)
se divide en subsecuencias del tipo: $A_i...A_{i+long}$ $\forall$ i, i=[1,n-*long*+1], donde *long* es el tamaño de las subsecuencias creadas y determina cuántos eventos se consideran como dependientes entre sí. En el resto de esta tesis, se utilizará el término *longitud de subsecuencia* para denotar este valor.

Considerando la secuencia de ejemplo propuesta: $\{A \rightarrow B \rightarrow A \rightarrow B \rightarrow C\}$ y una longitud de subsecuencia de tres elementos, se obtiene: $\{A \rightarrow B \rightarrow A\}$, $\{B \rightarrow A \rightarrow B\}$ y $\{A \rightarrow B \rightarrow C\}$.

2. Almacenamiento de las subsecuencias en el *trie*.

Partiendo de un *trie* vacío, cada una de las subsecuencias obtenidas a partir de la secuencia inicial se almacena en él de forma que todas estas subsecuencias

resulten accesibles. En un *trie*, cada evento se representa por un nodo, y sus descendientes representan los eventos que lo siguen. El camino del nodo raíz (*root*) a cualquier nodo, representa una lista de eventos ordenada.

En el enfoque que se propone, cada nodo del *trie* mantiene un contador con el número de veces que dicho evento se ha insertado en el nodo. Cuando una nueva subsecuencia se inserta en el *trie*, deberán modificarse aquellos nodos que aparecen en la subsecuencia o añadir nuevos nodos si éstos no existieran. Además, para resaltar la relevancia de las dependencias entre los eventos, se insertan también todos los sufijos de las subsecuencias. Por ejemplo, si la subsecuencia insertada es $\{A \rightarrow B \rightarrow C\}$, también se insertarán en el *trie* las subsecuencias $\{B \rightarrow C\}$ y $\{C\}$.

Si se consideran las 3 subsecuencias de ejemplo previas, $\{A \rightarrow B \rightarrow A\}$, $\{B \rightarrow A \rightarrow B\}$ y $\{A \rightarrow B \rightarrow C\}$, el proceso de inserción de dichas subsecuencias en una estructura *trie* vacía es el siguiente:

- La primera subsecuencia, $\{A \rightarrow B \rightarrow A\}$, se inserta en el *trie* como su primera rama (Figura 3.3 a). Cada nodo se etiqueta con el número 1 para indicar que el evento se ha insertado en el *trie* una vez. Este número se muestra en la figura entre corchetes.

- Los sufijos de la subsecuencia, $\{B \rightarrow A\}$ y $\{A\}$, también se insertan (Figura 3.3 b). De esta forma, se añade la nueva rama, $\{B \rightarrow A\}$, y se actualiza el nodo $\{A\}$, aumentando así en uno el número de elementos insertados en dicho nodo.

- Por último, una vez insertadas las tres subsecuencias y sus respectivos sufijos, el *trie* completo obtenido se puede observar en la Figura 3.3 c.

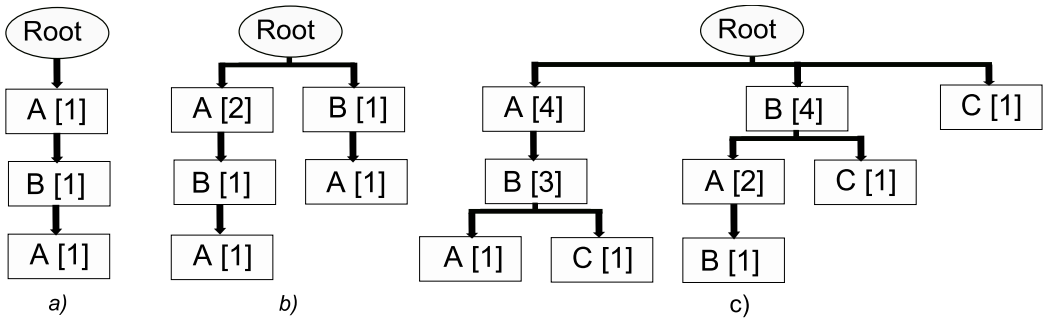


Figura 3.3: Pasos para la creación de un *trie*.

3. Cálculo de la relevancia de las subsecuencias.

Tabla 3.1: Tabla de contingencia χ^2

	Evento	Prefijo diferente	Total
Prefijo	O_{11}	O_{12}	$O_{11} + O_{12}$
Prefijo diferente	O_{21}	O_{22}	O_{21} + O_{22}
Total	O_{11} + O_{21}	O_{12} + O_{22}	O_{11} + O_{12} + O_{21} + O_{22}

Construido el *trie*, se deben determinar no sólo las subsecuencias más repetidos, sino las más relevantes. Para ello, se considera la dependencia que existe entre los nodos. En otras palabras, detectar qué caracteriza a las subsecuencias almacenadas en la estructura de datos. En este caso, se utilizan métodos de dependencia estadística [90].

Este tipo de métodos analizan la dependencia entre un evento y sus predecesores, Así, si un agente realiza siempre la acción *B* una vez finalizada la acción *A*, la dependencia entre ambas acciones será muy alta y por lo tanto es un dato significativo para el modelado de comportamiento de dicho agente.

Para analizar este tipo de dependencias entre elementos pueden utilizarse diferentes métodos. En este caso se propone el test de dependencia estadística [91], Chi-cuadrado (χ^2) [46]. Este test permite comparar objetivamente los eventos observados y esperados, para evaluar así la desviación que aparece entre ellos. Si se aplica este test al enfoque propuesto, cada evento almacenado en el *trie* tendrá asignado un valor que determine su relevancia con respecto a los eventos anteriores.

Para realizar el test χ^2, es necesaria una tabla de contingencia que represente la probabilidad de que dos elementos estén relacionados. Esta tabla almacena cuatro valores (Tabla 3.1) que son calculados del siguiente modo:

- O_{11}: indica cuántas veces se ha insertado dicho evento en el nodo.
- O_{12}: indica cuántas veces al prefijo del evento le sigue un evento diferente.
- O_{21}: indica cuántas veces a un prefijo diferente de la misma longitud le sigue el mismo evento.
- O_{22}: indica cuántas veces a un prefijo diferente de la misma longitud le sigue un evento diferente.

Los valores esperados se calculan mediante la ecuación 3.1.

$$Esperado(E_{ij}) = \frac{(Total\ Fila_i\ *\ Total\ Columna_j)}{Total} \tag{3.1}$$

Por último, el valor χ^2 se obtiene aplicando la ecuación 3.2.

$$X^2 = \sum_{i=1}^{2}\sum_{j=1}^{2}\frac{(O_{ij} - E_{ij})^2}{E_{ij}} \tag{3.2}$$

donde O_{ij} es la frecuencia observada y E_{ij} es la frecuencia esperada.

El valor χ^2 se calcula para cada uno de los eventos almacenados en el *trie* de forma que todos los eventos de la secuencia se identifican en función de sus eventos anteriores.

Si se añade este valor a cada uno de los nodos del ejemplo anterior (Figura 3.3) se obtiene el trie de la Figura 3.4 en el que el valor χ^2 se representa entre paréntesis. Como se puede observar, los eventos del primer nivel del *trie* no tienen un valor asociado debido a que éstos no tienen ningún elemento que les preceda.

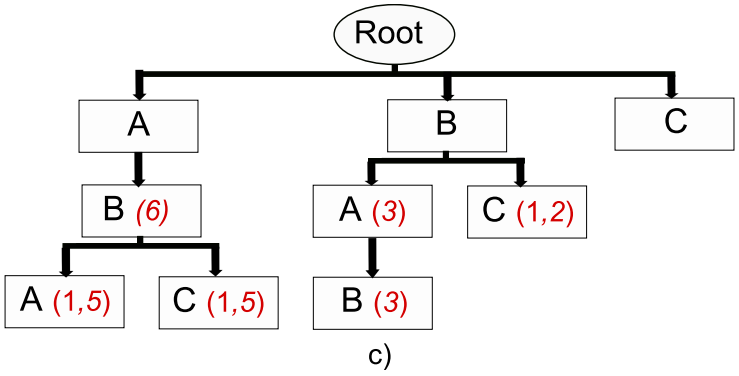


Figura 3.4: *Trie* que incluye el valor χ^2 en cada nodo.

3.3. Comparación de Comportamientos en modo lotes - MCCL

El proceso que se explica en este apartado puede utilizarse cuando, como ocurre en la competición *RoboCup Coach*, se desean comparar dos modelos de comportamientos para obtener sus diferencias más significativas.

Como se ha indicado en el subapartado anterior, todo comportamiento es modelado utilizando un *trie* cuyos nodos contienen la siguiente información:

- Evento: evento que se representa.
- Prefijo: conjunto de eventos ocurridos con anterioridad al evento que se considera.

- Valor χ^2: número que indica el valor χ^2 del evento.

La profundidad del *trie* es la profundidad máxima de todas sus ramas, y la raíz del trie es el nodo del nivel 0. Además, se debe definir un valor umbral que determine si se acepta o no la hipótesis del test χ^2 para cada evento.

Las principales características del algoritmo de comparación que se propone son:

- Si un evento se representa en ambos *tries* en el mismo nivel y con el mismo prefijo, se comparan sus valores χ^2. Si su diferencia es mayor que un umbral determinado *a priori* (*umbral_1*), dicho nodo junto con sus prefijos se consideran como parte del comportamiento que se busca. Es decir, se ha encontrado diferencia en los modelos de comportamiento que se corresponde con una parte del patrón de comportamiento que se busca.

- Si un evento junto con su prefijo están representados únicamente en el *trie* que representa el modelo con patrón, y el valor de su χ^2 es mayor que un umbral también determinado *a priori* (*umbral_2*), este nodo junto con sus prefijos se consideran como parte patrón de comportamiento que se busca.

3.3.1. Estructura básica del algoritmo

En este apartado se detalla la estructura básica del algoritmo de comparación. Para ello, se definen primero una serie de funciones:

- *ProfundidadTrie (Trie T):* Devuelve la profundidad máxima de las ramas de un *trie*.

- *GetCjtoDeNodos (NumeroEntero L, Trie T):* Devuelve el conjunto de todos aquellos nodos del *trie T* que se encuentren en el nivel L.

- *GetNodo (Evento E, Prefijo P, ConjuntoDeNodos S):* Devuelve del conjunto de nodos S aquel nodo cuyo evento es *E* y su prefijo *P*. Si no existe dicho nodo, esta función devuelve *null*.

- *Chi-2(nodo N):* Devuelve el valor χ^2 del nodo *N*.

- *Prefijo(nodo N):* Devuelve el prefijo (conjunto de eventos) del nodo *N*.

- *AddAlModelo(Evento E, Prefijo P, NumeroReal Valor_Chi-Cuadrado):* Añade al modelo, un evento determinado con su correspondiente prefijo y su valor χ^2.

Considerando estas funciones, la estructura básica del algoritmo tiene como entrada dos *tries* que representan dos comportamientos diferentes y como salida el patrón de comportamiento que se quiere modelar. Es decir, la diferencia entre los modelos con y sin patrón que también se representa en una estructura *trie* y se almacena en la biblioteca de modelos de comportamiento, BIBMOD. El algoritmo completo se muestra en la Figura 3.5.

3.4. Comparación de Comportamientos en Tiempo Real - MCTR

Almacenados los patrones comportamientos en BIBMOD, se debe detectar cuáles de dichos patrones siguen un conjunto de agentes mediante su observación en tiempo real. Para realizar esta tarea, hay que llevar a cabo los siguientes pasos:

1. Creación de un *trie* en tiempo real

   Mediante la observación del conjunto de agentes, se puede obtener la información necesaria para construir un *trie* que represente el comportamiento del conjunto de agentes. Dado que para modelar un comportamiento es necesario un número elevado de eventos, el *trie* se crea cada cierto tiempo en función de las características del dominio.

2. Comparación de *tries* en tiempo real

   El *trie* creado en tiempo real se compara con todos los *tries* almacenados en BIBMOD utilizando para ello un algoritmo de comparación diferente al utilizado para extraer el patrón de comportamiento. En este caso se buscan las similitudes entre ambos *tries* para determinar si existe o no un comportamiento similar en función de su valor χ^2. Además, se deberá considerar dicha similitud como parte del valor final que determina cuan similares son ambos *tries*. Así, este algoritmo utiliza las siguientes funciones:

   - *similitudNodos (Nodo nodo1, Nodo nodo2):* Devuelve si ambos nodos son o no similares en función de su valor χ^2. Es necesario establecer un umbral (*umbral_3*) que permita establecer esta diferencia.
   - *valorSimilitud (Nodo nodo1, Nodo nodo2):* Devuelve un valor que cuantifica la similitud entre ambos nodos, de forma que cuanto mayor sea este valor, mayor similitud existe entre los nodos.

   La Figura 3.6 muestra el algoritmo que compara dos *tries* en busca de sus similitudes.

Algoritmo de Comparación de Comportamientos - Búsqueda de diferencias

Entradas:
Trie que representa el comportamiento base (trie-Base)
Trie que representa el comportamiento a modelar (trie-Base+patrón)

Salida:
Modelo del patrón de comportamiento (trie-CompModelado)

Procedimiento:

```
CompararTries_BusquedaPatron (trie-Base, trie-Base+patron)

trie-CompModelado <-- null  //Comportamiento modelado obtenido como solución
maximaProfundidad <-- ProfundidadTrie (trie-Base)

// Raiz del nodo considerada como nivel 0. Eventos nivel 1 no tienen prefijos

FOR level_i:=2 TO maximaProfunidad DO

  cjto_C <-- GetCjtoDeNodos (level_i, trie-Base)
  cjto_NC <-- GetCjtoDeNodos (level_i, trie-Base+patron)

  FOR ALL nodo_C IN cjto_C DO

    node_NC <-- getNodo(Evento(nodo_C), Prefijo(nodo_C), cjto_NC)

    IF (nodo_NC = null)
    // El nodo está sólo en trie-Base

      IF (Chi-2(nodo_C) > Umbral_2)

        trie-CompModelado <-- AddAlModelo(Evento(nodo_C), Prefijo(nodo_C), Chi-2(nodo_C))

      END IF

    ELSE  //if node_NC != null)
    //El nodo está en ambos tries, se considera la diferencia de sus Chi-cuadrados

      IF (Chi-2(nodo_C) > Umbral_1 + Chi-2(nodo_NC))

        difChi2 = Chi-Cuadrado(node_C) - chi-2(node_NC)
        trie-CompModelado <-- AddAlModelo(Evento(nodo_C), Prefijo (nodo_C), difChi2)

      END IF

    END IF

  END FOR

END FOR

RETURN trie-CompModelado
```

Figura 3.5: Algoritmo comparación de *tries* para búsqueda de patrón de comportamiento

Algoritmo de Comparación de Comportamientos - Cálculo de similitud

Entradas:
Trie que representa el comportamiento observado (trie-observado)
Trie que representa el comportamiento almacenado (trie-almacenado)

Salida:
Valor que determina la similitud entre ambos tries

Procedimiento:

```
CompararTries_CalculoSimilitudes (trie-observado, trie-almacenado)

similitudTries <-- 0  //valor a devolver: similitud entre los dos tries.
maximaProfundidad <-- ProfundidadTrie (trie-observado)

// La raiz del nodo es considerada como nivel 0
// Los eventos en el nivel 1 no tienen prefijos

FOR level_i:=2 TO maximaProfunidad DO

  cjto_C1 <-- GetCjtoDeNodos (level_i, trie-observado)
  cjto_C2 <-- GetCjtoDeNodos (level_i, trie-almacenado)

  FOR ALL nodo_C1 IN cjto_C1 DO

    node_C2 <-- getNodo(Evento(nodo_C1), Prefijo(nodo_C1), cjto_C2)

    IF (nodo_C2 <> null)
    //El nodo está en ambos tries, puede existir cierta similitud

      IF similitudNodos(nodo_C1, nodo_C2)

         //Existe cierta similitud considerando el valor chi-cuadrado de ambos nodos.
         similtiudTries = similitudTries + valorSimilitud(nodo_C1, nodo_C2)

      END IF

    END IF

  END FOR

END FOR

RETURN similitudTries
```

Figura 3.6: Algoritmo de comparación de *tries* para calcular su similitud

3.5. Complejidad de la biblioteca BIBMOD

Cómo se ha comentado, cada uno de los modelos de comportamiento se construye utilizando un *trie* que se etiqueta para su posterior inserción en la BIBMOD. Como cada comportamiento se representa en la biblioteca con su correspondiente

estructura *trie*, el tipo de biblioteca utilizado en esta propuesta se ha denominado BIBMOD-K. Así, el número de *tries* depende del número de modelos creados, *k*. Sin embargo, la biblioteca también puede formarse con un único *trie* que almacene todos los modelos de comportamiento. Este tipo de bibliotecas se ha denominado BIBMOD-1. A continuación se explica cómo se construyen este tipo de bibliotecas.

En primer lugar, se crea un *trie* con la secuencia de eventos que representa el comportamiento del primer conjunto de agente. A continuación, y en este mismo *trie*, también se insertan las secuencias de eventos de otro conjunto de agentes. Por lo tanto, cada nodo del trie debe contener su evento asociado y un valor por cada uno de los comportamientos que representa. La inserción de una nueva subsecuencia en una BIBMOD-1 necesita considerar que dicha subsecuencia podría haber sido insertada previamente al representar otro comportamiento. Si se considera que *k* es el número de comportamientos que se quiere modelar, el nodo de un *trie* que almacena un evento en el que se representan *k* modelos, deberá ser *k* veces mayor que el nodo que almacena un evento que representa un único modelo.

La Figura 3.7 muestra los dos tipos de bibliotecas propuestos considerando como ejemplo dos comportamientos diferentes representados por las secuencias $\{A \rightarrow B \rightarrow A \rightarrow B \rightarrow C\}$ y $\{A \rightarrow B \rightarrow A \rightarrow D \rightarrow D\}$. En este caso los eventos de los nodos de la BIBMOD-1 tienen asociados dos valores, que se corresponden con los dos comportamientos que se quieren representar.

A pesar de que la información almacenada es la misma en ambas bibliotecas, el tiempo necesario para crear BIBMOD-1 y BIBMOD-K a partir de un mismo conjunto de secuencias, es muy diferente. Además, el tiempo para acceder a un determinado evento en cada una de las bibliotecas también varía. A continuación se estudia la complejidad de estos dos aspectos en las dos bibliotecas propuestas.

- **Creación de la biblioteca.**

  Dada la función *TiempoInsercion(n,l)* que devuelve el tiempo consumido para insertar *n* eventos en un *trie* usando subsecuencias de longitud *l* y asumiendo que todos los comportamientos que se desea insertar están representados por la misma cantidad de eventos; la ecuación 3.3 representa el tiempo consumido para crear BIBMOD-1 y la ecuación 3.4 representa el tiempo consumido para crear BIBMOD-K.

$$T(Crear\_BibMod\_k) = O(k * TiempoInsercion(n,l)) \tag{3.3}$$

$$T(Crear\_BibMod\_1) = O(TiempoInsercion(k * n,l)) \tag{3.4}$$

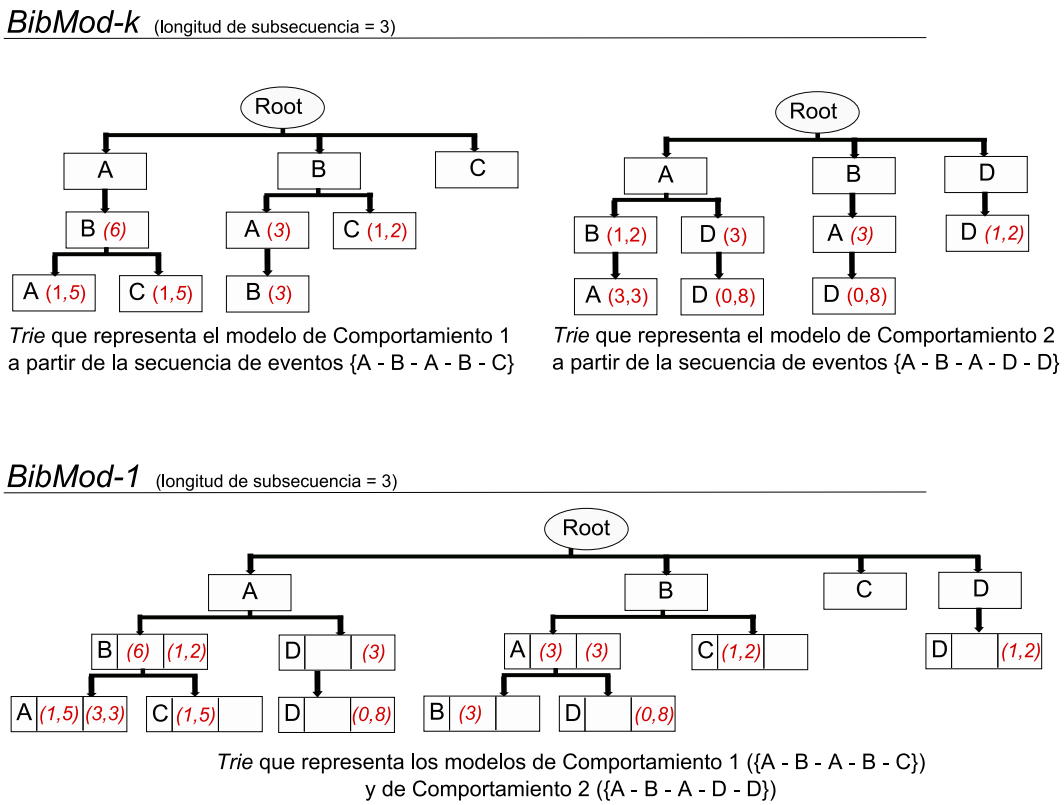


Figura 3.7: BIBMOD-1 y BIBMOD-K creados a partir de una misma secuencia de eventos y utilizando una longitud de subsecuencia de tres elementos.

donde n es el número de eventos por comportamiento que se van a insertar utilizando subsecuencias de longitud l, y k es el número de comportamientos diferentes que se modelan.

Para comparar estas dos ecuaciones, se debe tener en cuenta que el tiempo de inserción de un nuevo evento crece de forma exponencial en función del número de eventos insertados previamente. Así, la creación de una BIBMOD-1 consume más tiempo que la creación de una BIBMOD-K por su diferencia en el número de nodos que contiene.

- **Utilización de la biblioteca.**

  Si se aplica el enfoque propuesto en M-COMP, la biblioteca de comportamientos se crea una vez, mientras que la búsqueda en dicha biblioteca se realiza cada vez que se observa un nuevo comportamiento. De esta forma, si se desea localizar una subsecuencia de longitud l en una biblioteca de tipo BIBMOD-K, ésta se debe buscar en cada uno de los k *tries*. El tiempo

consumido, en el peor de los casos, se muestra en la ecuación 3.5.

$$T(BuscarEn\_BibMod\_k) = O(TiempoBusqueda(l) * k). \quad (3.5)$$

Si se realiza la misma búsqueda en una BIBMOD-1, el tiempo necesario será el tiempo de acceso al correspondiente nodo más el tiempo de acceso al evento correspondiente en dicho nodo. Así, la ecuación 3.6 muestra el tiempo consumido, en el peor de los casos, para realizar esta búsqueda.

$$T(BuscarEn\_BibMod\_1) = O(TiempoBusqueda(l) + k). \quad (3.6)$$

Por lo tanto, si se analizan estas dos ecuaciones, se observa que en este caso el tiempo de búsqueda de una subsecuencia es menor cuando se almacenan todos los comportamientos en un único *trie*, es decir en una biblioteca del tipo BIBMOD-1.

3.6. Evaluación: *RoboCup Coach*

En este apartado se explica de forma detallada la competición *RoboCup Coach 2006* y se describen los resultados obtenidos en este dominio utilizando M-COMP.

3.6.1. La RoboCup

La iniciativa *Robot World Cup* (*RoboCup*) [110] es un proyecto internacional que promueve la investigación en las áreas de la Inteligencia Artificial y la robótica. La competición de la *RoboCup* proporciona diferentes problemas estándar en los que pueden ser integradas y examinadas gran cantidad de técnicas utilizadas en la Inteligencia Artificial (IA). Así, dependiendo del objetivo buscado, este proyecto se divide en cuatro categorías: *RoboCupSoccer*, *RoboCupRescue*, *RoboCupJunior* y *RoboCup@Home*.

Esta experimentación se centra en la competición *RoboCupSoccer*, cuyo problema estándar consiste en que un conjunto de agentes sean capaces de jugar al fútbol. Así, el reto actual de esta competición es que para el año 2050 se haya desarrollado un equipo de robots humanoides totalmente autónomos capaces de vencer al campeón del mundo de fútbol de humanos. Este ambicioso objetivo necesita de muchos y muy diferentes aspectos a considerar. Por ello, actualmente esta competición se divide en 5 ligas en función de las características de los agentes participantes. La experimentación propuesta en este apartado se basa en la *Liga de Simulación 2D*.

Liga de Simulación

La Liga de Simulación 2D de la *RoboCup* proporciona un entorno complejo sobre el que se puede trabajar de forma sencilla y con unos medios limitados. Además, esta liga sólo utiliza aplicaciones software y no es necesaria ninguna infraestructura física, a parte de los propios ordenadores, para competir en ella. Estas características la han convertido en un entorno muy apropiado para el desarrollo de SMA.

Esta liga se basa en un servidor, denominado *SoccerServer* [145], que permite conectar dos equipos simulados formados por once agentes autónomos utilizando un protocolo de red estándar (Figura 3.8). Este servidor permite utilizar un agente *especial* denominado *coach* capaz de recibir toda la información del partido sin ruido y enviar información a su equipo utilizando un lenguaje específico. También se proporciona un entorno que visualiza el partido (Figura 3.9).

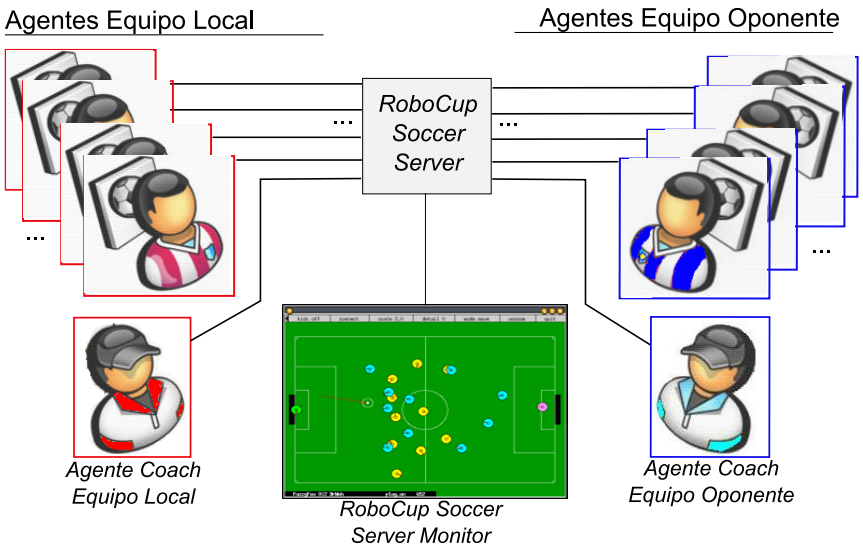


Figura 3.8: Esquema de la *Liga de Simulación 2D* de la *RoboCup*.

El servidor *SoccerServer* mantiene en todo momento el estado actual de los objetos en el entorno. Mediante una conexión UDP/IP, este servidor ejecuta las acciones solicitadas por los clientes (jugadores/agentes) añadiendo cierto ruido, y envía, periódicamente, a cada jugador información incompleta con ruido sobre el entorno. La información que recibe cada jugador es de tres tipos: visual, auditiva y sobre el estado de su cuerpo.

Entre la información visual que recibe el jugador se encuentra la posición del balón, de los jugadores, y de ciertas marcas situadas en el campo que ayudan a los jugadores a situarse dentro éste. Sin embargo, este tipo de información se recibe es relativa a su propia posición. Así, en lugar de recibir las coordenadas en las

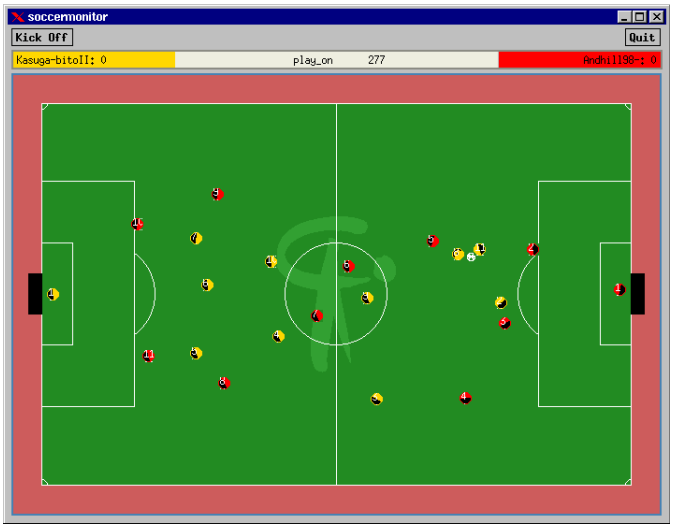


Figura 3.9: Monitor del sistema *Soccer Server*. Los agentes (jugadores) se representan como círculos y su color indica su equipo. La pelota se indica con el círculo blanco. Los agentes *coach* no están representados.

que se encuentra el elemento, el jugador obtiene la distancia y el ángulo a dicho elemento. Además, el ruido que se añade a esta información es proporcional a la distancia a la que se encuentre el elemento del jugador.

Las principales acciones primitivas que pueden realizar los jugadores son *dash* (correr), *turn* (girarse) y *kick* (chutar), las cuales se combinan para obtener acciones de alto nivel como pases o marcajes. Sin embargo, estas acciones no son deterministas ya que sobre ellas también se ejerce cierto ruido.

3.6.2. Competición *RoboCup Coach*

Dentro de la liga de simulación se disputa una sub-liga denominada *RoboCup Coach* y que en el resto del documento se denomina como competición *Coach*. En esta competición, además de los jugadores, cada equipo conecta al servidor el agente *coach* [45]. Este agente es similar al entrenador en el fútbol real y por lo tanto, no puede realizar acciones en el campo. En cada instante de tiempo, el *coach* recibe información sin ruido entre la que se encuentran las posiciones absolutas y las velocidades de cada uno de los jugadores y la pelota. Además, este agente es capaz de comunicarse con sus jugadores durante el partido utilizando un lenguaje estándar denominado *CLang* [45]. Así, el *coach* se convierte en una herramienta muy útil para el análisis táctico y el modelado del oponente [30]. Su utilidad se debe a que es capaz de tomar decisiones en función de su visión global del partido y comunicar a sus jugadores información relevante. La existencia del *coach* en

esta competición, hace que el dominio que trata sea muy adecuado para poder modelar el comportamiento de un conjunto de agentes, en este caso un equipo de fútbol simulado (el equipo oponente).

La primera competición *Coach* fue celebrada en el año 2001. En esta competición, el agente *coach* actuaba únicamente dando avisos o consejos a los jugadores [145] con la finalidad de estos mejoraran su comportamiento.

Uno de los primeros equipos en utilizar el agente *coach* en la *RoboCup* fue Kasugabito [176]. Este equipo utilizaba el *coach* para ajustar las formaciones del equipo basándose en aspectos como la diferencia del marcador, el tiempo transcurrido o el recorrido de la pelota. Por otra parte, Kuhlmann et al. [116] presentaron un agente *coach* capaz de construir un modelo de cada jugador oponente a partir de observaciones previas. Para ello, utilizan un clasificador por jugador, el cual trata de predecir qué hará dicho jugador en función de una situación determinada.

En el año 2005, las reglas de esta competición cambiaron para enfatizar la tarea de modelado de agentes. La principal finalidad de la competición ese año consistía en modelar y reconocer el comportamiento de otro equipo mediante la observación de su comportamiento. El comportamiento del equipo que se desea modelar no cambia en todo el partido, pero las condiciones dinámicas del entorno producen ciertas variaciones.

En el año 2006, la competición sufrió cambios leves y será la que se utilice como referencia en esta experimentación. Antes de explicar detalladamente esta competición, es necesario definir dos términos que se utilizan en ella pero que ya han sido introducidos al explicar el enfoque M-COMP:

- *Patrón de comportamiento:* Término que describe un comportamiento simple que afecta a una serie de jugadores. En esta experimentación, se utiliza el término *patrón* para referirnos en este entorno a un patrón de comportamiento. En este caso, un patrón no involucra a todos los jugadores del equipo, sino a un subconjunto de ellos.

- *Estrategia Base:* Estrategia general de un equipo de fútbol simulado en la que se pueden añadir patrones de comportamiento adicionales.

Los dos equipos que juegan un partido en esta competición se nombran como *Local* y *Oponente*. Cada participante de la competición deberá implementar el *coach* del equipo *Local*. A diferencia de otras competiciones, no importa el resultado ni lo bien que juegue el equipo *Local*, sino lo bien que el *coach local* modela, y posteriormente, detecta el comportamiento del *Oponente*.

El detalle de la competición *Coach 2006* [184] se muestra de forma gráfica en la Figura 3.10. Esta competición se divide en dos partes bien diferenciada, Modelado *Offline* y Detección *OnLine*, que se detallan a continuación.

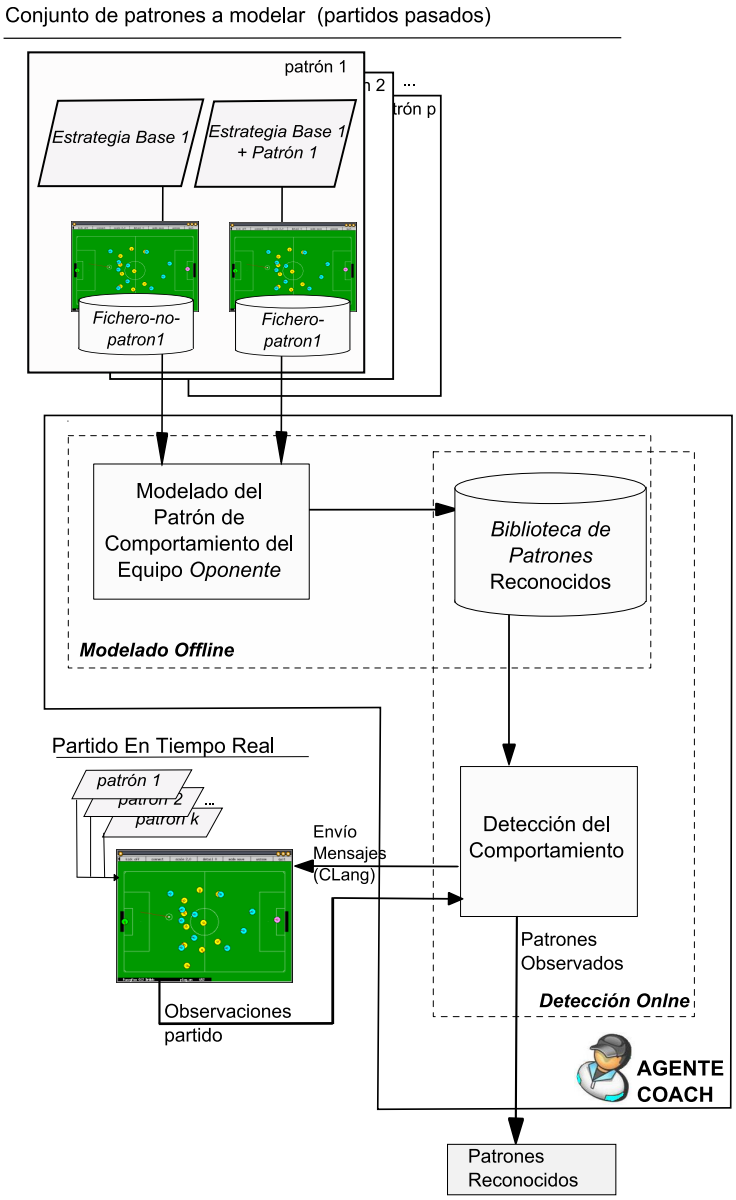


Figura 3.10: Estructura General de la competición *Coach*.

Modelado *Offline:*

- *Previo a esta fase*

  Antes de la competición, los organizadores deben definir qué comportamientos van a seguir los jugadores del equipo *Oponente*. Es decir, qué comportamientos debe modelar el *coach* del equipo *Local*. Para ello, se establecen varias estrategias base en *CLang* que indican los comportamientos que

deben seguir los jugadores del equipo *Oponente*. Existe un número determinado (p) de estrategias base. Con cada una de estas estrategias, se juega un partido y toda su información se almacena en un fichero denominado *fichero-no-patrón*. Así, se dispone de *p ficheros-no-patrón*. A continuación, utilizando el *CLang* se añade un patrón diferente a cada una de las p estrategias base y se juega un partido. La traza de este partido se almacena en un fichero denominado *fichero-patrón*. Así, obtendremos *p ficheros-no-patrón* con sus correspondientes p *ficheros-patrón*.

- *Finalidad de esta fase*

  Al comenzar la competición, el agente *coach* del equipo *Local* recibe los *p ficheros-patrón* junto con sus correspondientes *p ficheros-no-patrón*. A partir de estos ficheros, el *coach* debe modelar el patrón que se ha añadido a la estrategia base en cada uno de los *ficheros-comportamiento*. Respecto a cómo se almacena cada uno de estos comportamientos, no hay ninguna restricción y cada participante puede utilizar su propia estructura.

  Una vez que se han detectado los p patrones propuestos, éstos se almacenan en la *Biblioteca de Patrones Reconocidos*. Esta biblioteca tiene también un formato libre.

Detección *Online*:

- *Previo a esta fase:*

  Para llevar a cabo esta segunda fase, los organizadores implementan un equipo que sigue algunos de los patrones de comportamiento que han sido analizados en la fase anterior. Para ello, se selecciona un número variable de patrones de comportamiento (k), y se implementa un equipo. Esta selección de k patrones de comportamiento para un mismo equipo se hace de forma que cada jugador sólo siga un único patrón. Por ejemplo, si se define el *patrón A* como *el portero siempre pasa al jugador 3* y el *comportamiento B* implica *el portero siempre pasa al jugador 2*, obviamente, estos dos patrones no pueden agruparse en un único *comportamiento de equipo*.

- *Finalidad de esta fase:*

  En esta fase, el equipo *Local* se enfrenta en tiempo real al equipo *Oponente* el cual sigue los k patrones seleccionados entre los p patrones analizados. El *coach* del equipo *Local* debe detectar los diferentes patrones que se han activado en el equipo *Oponente* durante el partido y reportar su detección lo antes posible. Para ello, deberá utilizar la *Biblioteca de Patrones Reconocidos*. Cuanto antes detecte un patrón, mayor puntuación se obtendrá. Pero

si el comportamiento reportado es incorrecto, es decir, no corresponde con uno de los patrones que está siguiendo el equipo *Oponente*, se recibe una penalización. Además, el *coach* puede enviar mensajes a los jugadores de su equipo (*Local*) para detectar mejor los patrones. En esta competición, sólo es necesario reportar el nombre del patrón, no el comportamiento que éste patrón implica. Por lo tanto, no se puntúa en función de lo bien que un patrón es modelado, sino de lo bien que éste es reconocido.

3.6.3. Aplicación de M-COMP

En este apartado se detalla cómo se aplica cada uno de los módulos de M-COMP en el dominio propuesto.

A. Modelado Offline

A.1. Obtención de la secuencia de eventos futbolísticos (MCSE).

El primero paso para aplicar M-COMP consiste en la creación del secuencias a partir de observaciones. Es decir, el proceso incluido en MCSE. En este caso, dicha secuencia está formada por los eventos futbolísticos que definen el comportamiento de un equipo. Un equipo de fútbol realiza en un partido muchas y diversas acciones. Los eventos son aquellas acciones que *a priori* se consideran relevantes para el modelado del equipo y por lo tanto, se quieren detectar en el juego. Esta secuencia de eventos se obtiene a partir del fichero binario que almacena lo sucedido en un partido. Este fichero permite conocer el estado del partido en cada instante de tiempo. A continuación, se explica cómo se realiza este proceso.

El servidor de simulación de fútbol 2D de la *RoboCup* (*Soccer Server*) almacena todos los datos de un partido en dos ficheros diferentes:

- Un fichero de texto en el que se almacenan todos los comandos de comunicación entre los jugadores/clientes y el servidor.

- Un fichero binario que almacena lo sucedido en un partido de forma que sea posible reproducirlo posteriormente. Este fichero almacena información general del partido, como por ejemplo, parámetros del servidor, tamaño del jugador y de la pelota, distribución del área de regate de un jugador y nombre de los equipos. Además, por cada uno de los 6.000 ciclos de los que consta un partido, se almacenan los siguientes datos: número de ciclo, modo de juego (modo normal, fuera de juego, gol, saque de puerta, saque de esquina, saque de centro y saque de banda), posición y velocidad de la pelota y de cada jugador, número que indica quién es el poseedor de la pelota, ángulo del cuerpo de cada jugador, número de veces que cada jugador ha pateado la pelota y número de veces que el portero ha cogido la pelota.

En la competición *Coach*, el procedimiento de identificación de eventos futbolísticos se debe realizar únicamente a partir del fichero binario. La base de este procedimiento es un trabajo realizado por Kaminka et al. [104] y ampliado por Kuhlmann et al. [116] en el que, inicialmente, se obtienen de cada uno de los 6.000 ciclos del partido los siguientes datos:

- Ciclo del partido.
- Posición y velocidad de la pelota.
- Posición y velocidad de todos los jugadores.
- Poseedor de la pelota.

A partir de estos datos, se obtienen los eventos que ha realizado el equipo en función de dónde se encuentra la pelota en cada momento. En este caso se consideran únicamente aquellas acciones relacionadas con la posesión de la pelota. Las acciones en las que la pelota no intervenga, no se analizan. Se considera que todo partido está formado por una serie de posesiones sucesivas y que un cambio de posesión ocurre cuando el balón cambia de dueño, se marca un gol o el modo de juego cambia (por falta, fuera de juego, etcétera). Así, cada posesión está formada por dos partes:

- La primera parte se denomina *intervalo posesión (IntPos)* y comienza cuando el jugador se hace con la pelota, prolongándose hasta que ésta se aleje del rango de posesión del jugador.
- La segunda parte se denomina *intervalo libre (IntLibre)* y comienza en el ciclo siguiente al último ciclo del *IntPos* acabando en el siguiente cambio de posesión.

A partir de la identificación de estas partes en el juego, se definen ocho eventos futbolísticos que se detallan a continuación:

- *Regate:* La pelota se mueve una distancia significativa durante el *IntPos*.
- *Mantener la pelota:* Un jugador mantiene inmóvil el balón durante varios ciclos.
- *Gol:* En el último ciclo de *IntLibre* el balón se encuentra dentro de la portería.
- *Pase:* En el último ciclo de *IntLibre* un jugador del mismo equipo que el jugador poseedor del balón durante el anterior *IntPos*, recupera el balón.

- *Falta:* El modo del juego en el último ciclo de *IntLibre* señala una falta. No se consideran distintos tipos de falta.
- *Robo de la pelota:* La pelota cambia de dirección pero está a una distancia corta del poseedor del balón.
- *Tiro fuera:* En el último ciclo de *IntLibre* el balón se encuentra cerca de la portería (pero no dentro) y al comienzo de dicho intervalo se encontraba relativamente cerca de ella.
- *Interceptación de pase:* En el último ciclo de *IntLibre* un jugador del equipo contrario que se encuentra a una distancia determinada, recupera el balón.

El resultado del MCSE en este caso, es una secuencia con todos los eventos ocurridos durante el partido. Estos eventos están ordenados en función del tiempo. Por otra parte, hay que tener en cuenta que:

- Los eventos futbolísticos que se obtienen son los que se consideran más representativos y a partir de los cuales se generará el modelo de comportamiento de un equipo. Sin embargo, estos eventos se pueden modificar o incluso añadir otros eventos nuevos obteniendo así una secuencia diferente.
- Los eventos utilizados sólo consideran las acciones realizadas por los jugadores cuando éstos poseen el balón. Sin embargo, el comportamiento de un equipo podría hacer referencia a situaciones imposibles de reconocer con los eventos utilizados. Por ejemplo, un comportamiento podría estar relacionado con la posición de un jugador sin que éste sea poseedor del balón.
- Se pueden detectar acciones no intencionadas de un jugador. Por ejemplo, un jugador podría tirar a puerta, pero si el tiro lo intercepta un compañero, se considera que ha realizado un pase.
- Existen parámetros configurables en el proceso de detección de eventos. Por ejemplo, cuándo es *significativa* una distancia para que la acción sea considerada como un regate.

Todo evento está identificado con los jugadores involucrados en él y el equipo al que pertenecen estos jugadores. Así, parte de una secuencia de eventos de un partido de fútbol podría tener el siguiente aspecto:

$\{Pase1A2\ (D) \rightarrow Pase2A10\ (D) \rightarrow Gol10\ (D) \rightarrow Pass7A10\ (I) \rightarrow TiroFuera10\ (I) \rightarrow Pase1A2\ (D) \rightarrow Regate2\ (D)\}$.

Se observa que cada evento tiene uno o dos números que representan los jugadores involucrados en dicha acción y una letra (D (derecha), I (izquierda)) indicando el equipo del jugador que ha realizado el evento. Por ejemplo el evento «*Pase1A2 (D)*» indica que el jugador 1 del equipo derecho ha realizado un pase al jugador 2 de su mismo equipo.

A pesar de que los eventos futbolísticos que se obtienen corresponden a los dos equipos involucrados en el partido; sólo se modela el comportamiento del equipo *Oponente*. Por esta razón, sólo se consideran los eventos realizados por el equipo que se desea modelar. Si se considera el ejemplo anterior, y el equipo que se desea modelar es el *Derecho*, se obtendrán secuencias:

$\{Pase1A2\ (D) \rightarrow Pase2A10\ (D) \rightarrow Gol10\ (D)\}$ y
$\{Pase1A2\ (D) \rightarrow Regate2\ (D)\}$

Dada las subsecuencias que se obtienen al dividir los eventos en función del equipo que haya realizado la acción, no es necesario definir una *longitud de subsecuencia*. La longitud de subsecuencia está determinada por la posesión del balón.

A.2. Creación del modelo de comportamiento (MCMC):

Lo primero es crear los dos modelos de comportamiento del equipo *Oponente* en cada uno de los partidos previos utilizando el MCMC. El modelo con patrón y el modelo sin patrón se comparan utilizando el MCCL y se obtiene el modelo del patrón que se ha añadido a la estrategia base del equipo *Oponente*, el cuál se almacena en BIBMOD. La duración máxima de cada partido es de 6.000 ciclos, aproximadamente diez minutos de juego. Este proceso se realiza tantas veces como patrones se necesiten modelar.

B. Detección *OnLine:*

Una vez que el agente *Coach* ha analizado los comportamientos del equipo *Oponente* en partidos previos y ha almacenado los patrones específicos en BIBMOD, se debe detectar qué patrones sigue el equipo oponente en un partido observado en tiempo real. Para realizar esta tarea, el agente *coach* utiliza el MCSE, el MCMC y el MCTR.

B.1. Creación del modelo de comportamiento en tiempo real (MCSE y MCMC)

Conectado el agente *Coach* al servidor *RoboCup Soccer Server*, éste recibe información sobre todos los datos del entorno. La información que se recibe es

almacenada y utilizando el MCSE se obtiene la secuencia de eventos correspondiente cada 1500 ciclos. Esta secuencia de eventos es utilizada por el MCMC para crear el modelo de forma que los modelo construidos cada 1500 ciclos son significativamente diferentes entre sí. Como un partido tiene 6.000 ciclos, se creará un modelo de comportamiento a los 1500 ciclos de partido y éste se actualizará tres veces durante el partido.

B.2. Comparación de *tries* en tiempo real:

A los 1500 ciclos, el modelo creado utilizando MCMC se compara con todos los modelos de los patrones almacenados en BIBMOD. En este caso, si la similitud entre ambos modelos es mayor que un umbral, se determina que el comportamiento del equipo observado en tiempo real sigue dicho patrón. Dado que cuanto antes se detecte el patrón, mayor puntuación se obtiene en la competición; se realizan otras tres comparaciones a los 2.950, 4.500 y 5.950 ciclos.

3.6.4. Configuración Experimental

La competición *Coach 2006* estaba dividida en tres rondas. Sin embargo, la tercera ronda compartía características con las dos primeras. Por esta razón, sólo se consideran los datos de las rondas uno y dos. Además, dada la importancia que tiene el número de veces que un patrón se ejecuta, se ha realizado un análisis detallado en la etapa de *Detección online*.

Modelado de los patrones - *Modelado Offline*

En cada una de las rondas, el *coach* de cada equipo participante debía analizar varios comportamientos. Así, cada equipo recibe varios *ficheros-patrón* con su correspondiente *fichero-no-patrón* y el tiempo máximo para analizar y modelar el patrón de un equipo era de cinco minutos.

La descripción de todos los patrones que se utilizaron en esta competición se detalla en el apartado A.1 del apéndice A. La cantidad de patrones que se consideraron fueron 17 en la primera ronda (nombrados del 0 al 16) y 16 en la segunda (nombrados con letras de la A a la P).

Detección de patrones - *Detección Online*

En la fase de *Detección online*, cada ronda se dividió en tres iteraciones y en cada una de ellas el *coach* de un equipo observaba en tiempo real al equipo *Oponente*, en el cuál se habían *activado* cuatro o cinco patrones. Los comportamientos activos variaban en cada iteración. Para implementar un equipo con estos

patrones, era necesario que dos o más patrones no afectaran a un mismo jugador para evitar ambigüedades. Este factor se toma en cuenta al seleccionar los patrones que se activan en cada una de las tres iteraciones de cada ronda. Los patrones seleccionados para cada iteración de cada ronda se muestran en la Tabla 3.2.

Tabla 3.2: Comportamientos utilizados en la ronda 1 y 2 de la competición *Coach 2006*

Ronda 1			
Patrones de comportamiento (***Off-Line***)	Patrones de comportamiento *activados* en el equipo oponente (***On-Line***)		
	Iteración 1	Iteración 2	Iteración 3
patrón_00 *patrón_01* *patrón_02* *patrón_03* *patrón_04* *patrón_05* ... *patrón_15* *patrón_16*	*patrón_00* *patrón_04* *patrón_14* *patrón_15*	*patrón_01* *patrón_07* *patrón_08* *patrón_13* *patrón_16*	*patrón_00* *patrón_02* *patrón_04* *patrón_06* *patrón_12*
Ronda 2			
Patrones de comportamiento (***Off-Line***)	Patrones de comportamiento *activados* en en el equipo oponente (***On-Line***)		
	Iteración 1	Iteración 2	Iteración 3
patrón_A *patrón_B* *patrón_C* *patrón_D* *patrón_E* *patrón_F* ... *patrón_P* *patrón_Q*	*patrón_A* *patrón_F* *patrón_H* *patrón_J* *patrón_O*	*patrón_E* *patrón_I* *patrón_K* *patrón_P*	*patrón_B* *patrón_E* *patrón_I* *patrón_M*

Al implementar varios patrones en un único equipo, es posible que ciertos patrones no se ejecuten o lo hagan muy pocas veces. Las Figuras 3.11 y 3.12 muestran el número de veces que un jugador del equipo oponente ha ejecutado cada patrón activo en las iteraciones de cada ronda. Estas gráficas muestran por

cada patrón activo:

- Número de veces que un jugador ha podido ejecutar el patrón.
- Número de veces que un jugador no ejecuta dicho patrón porque otro jugador interviene.
- Número de veces que un jugador ejecuta un comportamiento diferente al que debe realizar.

Si se considera como ejemplo el patrón nombrado como *patrón_15* y detallado en la Figura 3.14, se observa que los jugadores 1, 2 o 10 han ejecutado el patrón correctamente 19 veces. Sin embargo, en 26 ocasiones ha intervenido otro jugador en el pase. En 4 ocasiones los jugadores han ejecutado una acción diferente. En el apartado A.2 del apéndice A se puede encontrar más detalles sobre el análisis de los comportamientos activos.

3.6.5. Resultados

M-COMP fue desarrollado para la competición *Coach 2006* [184] como parte del *CAOS Coach Team* [94]. En el apéndice A se detallan los resultados, equipos y puntuaciones de la competición.

En este apartado se muestran todos los patrones reportados al final del partido por M-COMP. Es importante señalar que los umbrales necesarios al momento de aplicar M-COMP se han determinado de manera experimental. La Tabla 3.3 muestra estos valores.

Tabla 3.3: Umbrales utilizados por M-COMP en la competición *Coach 2006*

Nombre	Valor	Utilizado en	Descripción
umbral_1	20	Algoritmo del MCCL	Umbral para considerar como diferencia de comportamiento una misma secuencia de eventos almacenada en los dos *tries*.
umbral_2	60	Algoritmo del MCCL	Umbral para considerar una secuencia de eventos almacenada en el *trie* del modelo con patrón como parte del comportamiento que se desea buscar.
umbral_3	15	Algoritmo del MCTR	Umbral para considerar si dos secuencias de eventos de dos *tries* diferentes son similares.

La Figura 3.13 muestra los patrones activos en cada iteración y ronda en función del número de veces que realmente se ejecuta cada patrón. Remarcados mediante un rectángulo rojo se muestran los patrones detectados correctamente por

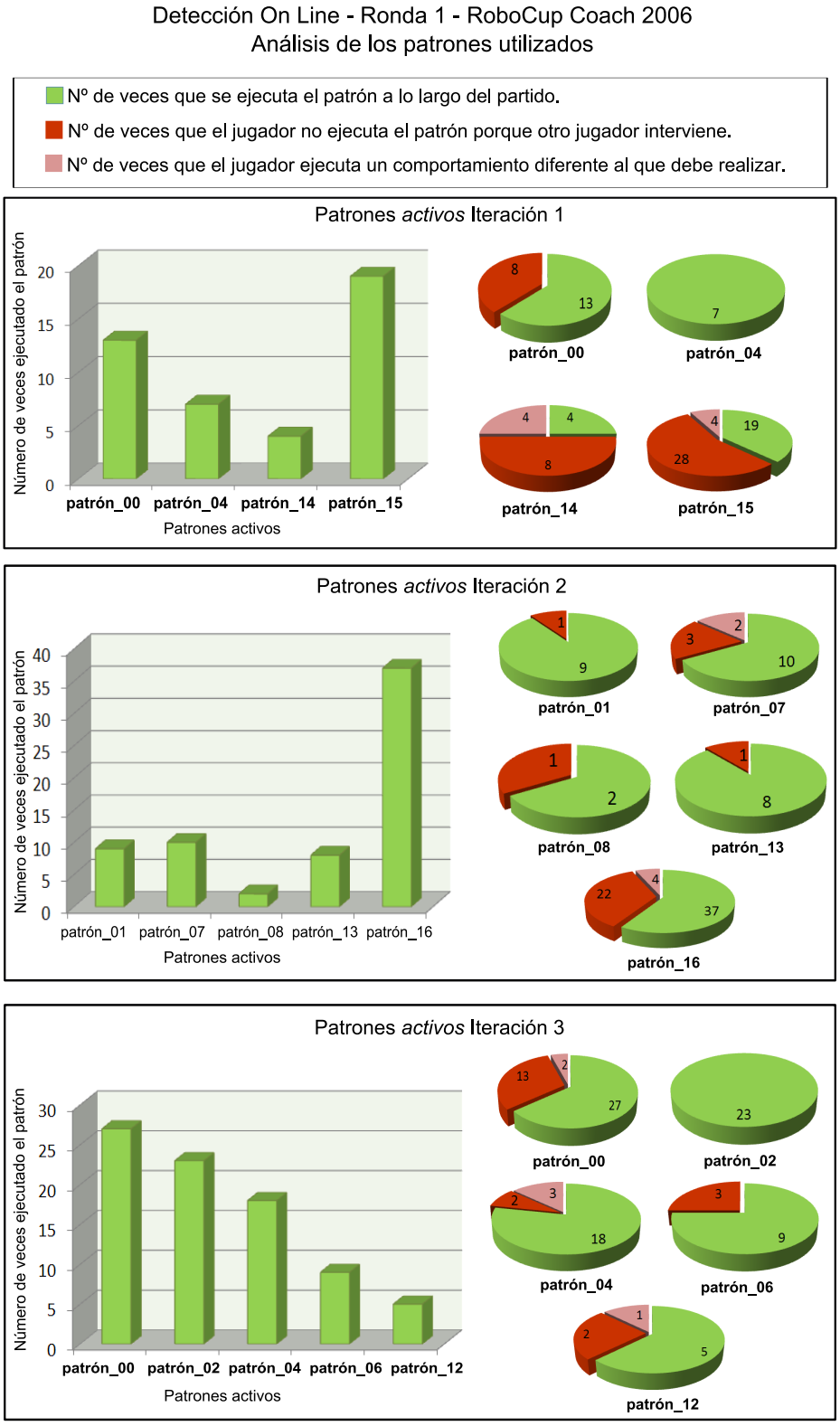


Figura 3.11: Análisis de Comportamientos - Detección online. *Coach 2006* - Ronda 1

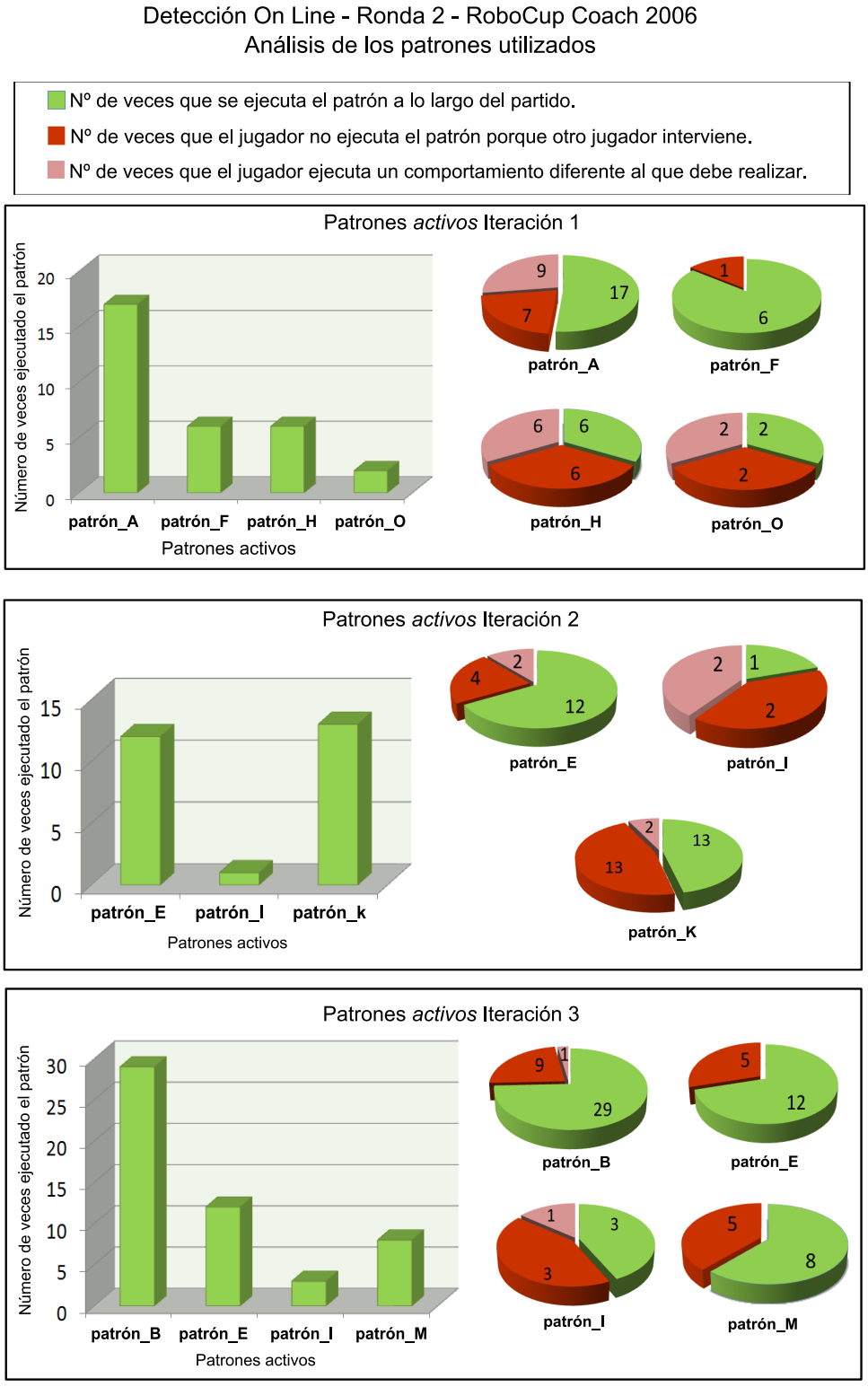


Figura 3.12: Análisis de Comportamientos - Detección online. *Coach 2006* - Ronda 2

M-COMP. Los patrones que no se detectaron correctamente no se muestran. Sin embargo, en cada iteración se reportaban tantos patrones como número de éstos se habían activado en el equipo *Oponente*. Por ejemplo, en la primera iteración de la primera ronda se reportan 2 patrones correctos y 2 erróneos, mientras que en la segunda iteración, se reportan 2 correctos y 3 erróneos.

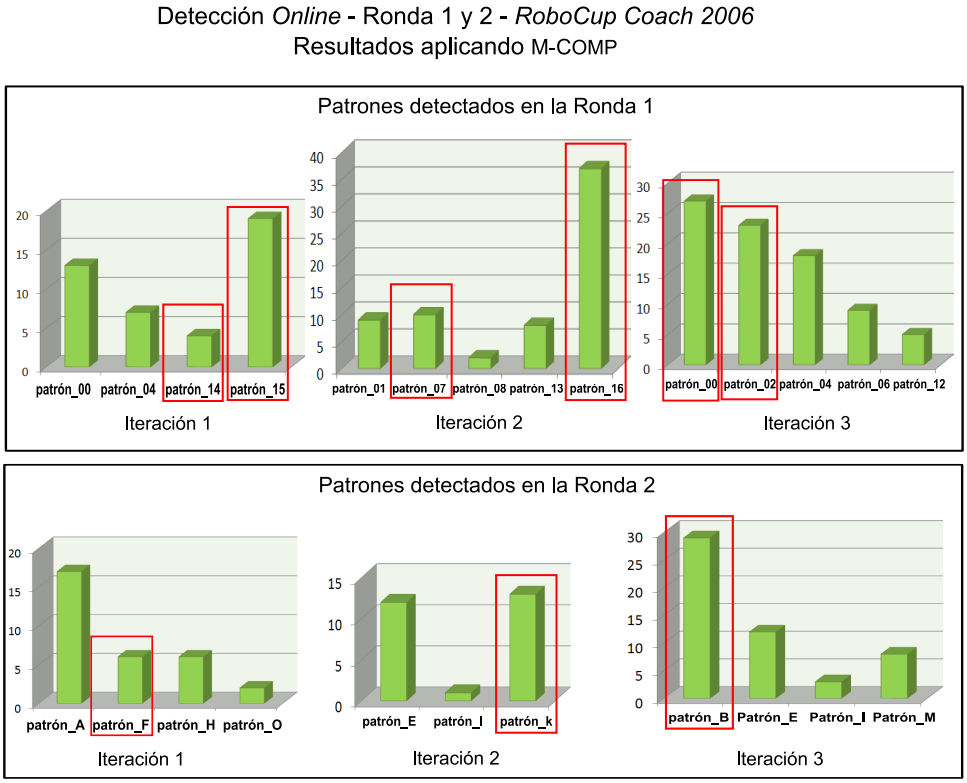


Figura 3.13: Resultados - Detección online. *Coach 2006* - Rondas 1 y 2

La Figura 3.14 muestra la descripción en *CLang* y en lenguaje natural del *patrón_15*. En este caso, se observa que los jugadores involucrados en el patrón son el 1, el 2 y el 10. Cada uno de estos jugadores deberá realizar la acción *pase* a un punto diferente del campo. También se muestran las secuencias de eventos futbolísticos más relevantes almacenadas en el modelo del patrón obtenido por M-COMP. Se observa que las dos primeras acciones descritas en el patrón son representadas en el modelo mediante la asignación de un elevado χ^2 a la subsecuencia de eventos correspondiente. Además, el patrón incluye ciertos eventos que no forman parte del patrón. Este patrón es reconocido en la competición.

Analizando los resultados, M-COMP es capaz de reconocer, generalmente, los patrones relacionados con los eventos futbolísticos definidos. En el apartado A.3 del apéndice A se muestra un análisis del tipo de acciones consideradas en

patrón_15

CLang:	Descripción:
(say (define (definerule RULEPASS12 direc ((bowner our {1}) (do our {1} (pass {2})))))	El portero siempre pasará al jugador 2.
(say (define (definerule RULEPASS210 direc ((bowner our {2}) (do our {2} (pass {10})))))	El jugador 2 pasará al jugador 10.
(say (define (definerule RULEDRIBBLE10 direc ((and (playm play_on)(bowner our {10})) (do our {10} (pass ((pt our 10)+(pt 0 20))))))))	El jugador 10 siempre lanzará la pelota a un punto determinado del campo.

Eventos más relevantes representados en el modelo del patrón reconocido por M-Comp:

Evento	Evento/s anterior/es	χ^2
Jugador 2 pasa al jugador 10	Portero pasa al jugador 2	72.0
Jugador 2 pasa al jugador 7	Portero pasa al jugador 2	45.0
Jugador 7 pasa al jugador 10	Jugador 2 pasa al Jugador 7	36.0
Jugador 10 pasa al portero	Jugador 2 pasa al Jugador 10 Jugador 2 intercepta la pelota	31.0
Jugador 9 regatea	Jugador 10 pasa al jugador 9	31.0

Figura 3.14: *Patrón_15*. Reglas *CLang* y descripción.

los patrones y si éstas son modeladas correctamente por M-COMP. Este análisis muestra que existen determinados patrones en los que los eventos relevantes del modelo creado por M-COMP, no se corresponden con las acciones que definen dicho patrón. Sin embargo, el modelo obtenido está relacionado con ciertas acciones provocadas por la ejecución de dicho patrón. Por ejemplo, en el *patrón_02*, los jugadores 9, 10 y 11 siempre pasan la pelota a un determinado punto. Sin embargo, en el modelo obtenido por M-COMP son relevantes las acciones en las que el jugador 9 pasa al 3, y el jugador 1 pasa al 6. En este caso, a pesar de no haberse obtenido un modelo tal y como éste se definió, el patrón es capaz de reconocerse.

Además, si el número de veces que se ejecuta un patrón es escaso, M-COMP no es capaz de detectarlo. Por ejemplo el *patrón_O* de la iteración 1 de la ronda 2, es definido correctamente en el modelo creado. Sin embargo, se ejecuta pocas veces por el equipo *Oponente*, por lo que no es posible detectarlo.

Los resultados también demuestran que el problema planteado en la competición *Coach 2006* es complejo. Una regla de la competición es que todo patrón debía ser predecible y repetirse varias veces. Sin embargo, el concepto “predeci-

ble" es algo confuso y el número de veces que debe repetirse un comportamiento, es también cuestionable. Esta característica hace que sea difícil valorar tanto el enfoque propuesto como los resultados obtenidos de forma concisa. Además, debido a la diversidad de comportamientos posibles en este entorno y lo diferentes que pueden ser unos de otros, quizás la combinación de diferentes técnicas para distintos tipos de comportamientos sea la mejor forma de afrontar este problema.

3.7. Conclusiones

En este capítulo, se ha presentado el primero de los enfoques de esta tesis doctoral. Este enfoque fue desarrollado para la competición *Coach 2006*. Sin embargo, M-COMP puede aplicarse en dominios con características similares a la competición. Es decir, modelar y comparar comportamientos para detectar patrones específicos y ser capaz de detectar dichos patrones en tiempo real.

En relación a la competición se debe resaltar lo complejo que resulta reconocer un determinado patrón de comportamiento en este entorno. En este tipo de dominio, el comportamiento de un agente varía en función del resto de jugadores. Un entorno de estas características y que requiera reconocer los patrones casi en tiempo real, hace que esta detección sea aún más compleja. Las interacciones entre agentes producen nuevos comportamientos que son difíciles de predecir por la gran cantidad de variables que intervienen.

A pesar de la dificultad del problema, los resultados obtenidos por M-COMP, muestran que éste es adecuado para detectar cierto tipo de comportamientos. Debido a que M-COMP transforma las observaciones en secuencias de eventos, la definición de los eventos es un factor fundamental en el resultado. Si, como en este caso, dichos eventos sólo hacen referencia a las acciones realizadas cuando los jugadores tienen el balón, sólo es posible reconocer este tipo de patrones.

Capítulo 4

Modelado de Agentes Utilizando Secuenciación de Eventos: MAUSE

En este capítulo se presenta el segundo de los tres enfoques propuestos en esta tesis doctoral denominado MAUSE (*Modelado de Agentes Utilizando Secuenciación de Eventos*). Este enfoque tiene como finalidad modelar y detectar el comportamiento de uno o varios agentes a partir de una secuencia ordenada de eventos que definen su comportamiento. A diferencia de M-COMP, MAUSE modela el comportamiento a partir de una única secuencia de eventos.

La Figura 4.1 muestra los componentes del MAUSE que, al igual que M-COMP, está implementado en un agente:

1. Al igual que en M-COMP la primera parte de MAUSE consiste en la obtención de secuencias de eventos a partir de la observación del comportamiento de un conjunto de agentes. Esta conversión es realizada en MCSE, *Módulo de Creación de Secuencias de Eventos*, detallado en el apartado 3.1.

2. A continuación, se construye el modelo de comportamiento de uno o varios agentes a partir de la secuencia de eventos obtenidos previamente. El modelo lo crea el *Módulo de Creación de Modelos de Comportamiento*, MCMC (apartado 4.1). A diferencia del MCMC utilizado en M-COMP, el MCMC de este enfoque utiliza métodos basados en frecuencia para calcular la relevancia de las subsecuencias que forman dicho modelo.

3. En una segunda fase, se detecta, en tiempo real, el comportamiento de un agente o conjunto de agentes a partir de su observación utilizando los modelos almacenados en BIBMOD. Para ello, se propone un clasificador basado en métodos estadísticos. En el apartado 4.2 se explica de forma detallada el funcionamiento de este clasificador.

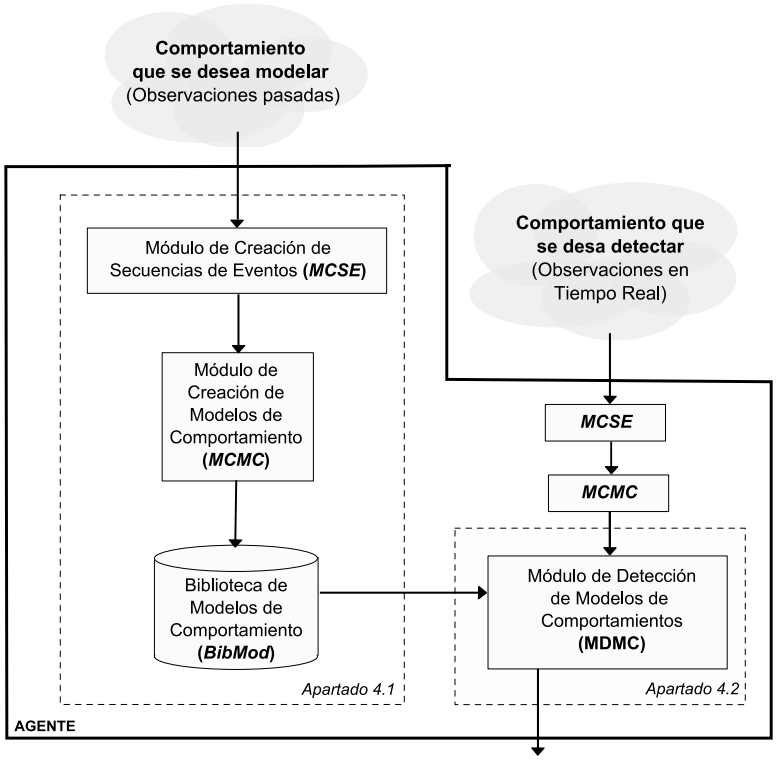


Figura 4.1: Estructura general de MAUSE - Modelado de Agentes Utilizando Secuenciación de Eventos.

4.1. Creación y almacenamiento de Modelos de Comportamiento

Este método se basa en el propuesto en el apartado 3.2 y parte de una serie de observaciones de las acciones realizadas por el conjunto de agentes que se desea modelar. Al igual que M-COMP, este proceso tiene tres componentes, el MCSE, el MCMC y BIBMOD. Sin embargo, en MAUSE alguno de estos módulos han sido modificados.

4.1.1. Obtención de Secuencias de Eventos - MCSE

Este módulo, al igual que el propuesto en el capítulo anterior, recibe como entrada las observaciones del agente o conjunto de agentes que se desea modelar y se obtiene la secuencia de eventos correspondiente. La obtención de estas secuencias depende del dominio e influye en gran medida en los resultados que se obtienen.

4.1.2. Creación del Modelo de Comportamiento - MCMC

Al igual que en M-COMP, el MCMC utiliza una estructura *trie* para el almacenamiento de las subsecuencias de comportamiento generadas por MCSE como primer paso para la obtención del modelo de comportamiento. Sin embargo, a la hora de calcular la relevancia de los eventos, se utilizan métodos basados en frecuencia. Esté cálculo se detalla en el siguiente apartado.

Cálculo de la relevancia de los eventos

Construido el *trie*, se deben determinar no sólo los eventos más repetidos, sino los más relevantes en la secuencia. Para este cálculo, el MCMC calcula la importancia de una determinada subsecuencia de eventos utilizando métodos basados en frecuencia por sus buenos resultados en la minería de secuencias de datos [3]. Cómo calcular este valor y almacenarlo en cada nodo se detalla a continuación.

Mediante los métodos basados en frecuencia se obtiene la relevancia de una subsecuencia de eventos en función del número de veces que ésta ha sido insertada en el *trie*. El MCMC utiliza la *frecuencia relativa* [3] para obtener las subsecuencias relevantes. En este caso, se define la *frecuencia relativa* de una subsecuencia como el número de veces que la subsecuencia se ha insertado en el *trie* en función del número total de subsecuencias insertadas de la misma longitud. Es decir, el número total de subsecuencias insertadas en el mismo nivel. Con la finalidad de comparar eventos de las mismas características, es importante realizar este cálculo considerando únicamente subsecuencias de la misma longitud. Este valor se almacena en los nodos del *trie* correspondiente.

Si se considera el ejemplo del capítulo anterior, en el *trie* (Figura 3.3) se deben añadir los valores de *frecuencia relativa* de cada uno de los nodos. La Figura 4.2 muestra esta modificación mostrando la *frecuencia relativa* entre paréntesis.

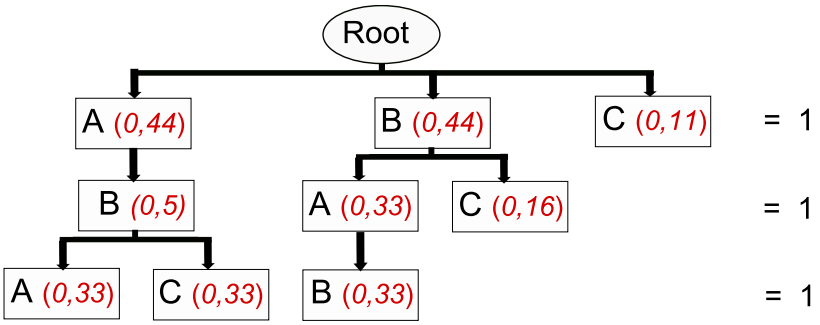


Figura 4.2: *Trie* que incluye el valor *frecuencia relativa* en cada nodo.

4.1.3. Almacenamiento de Modelos de Comportamiento - BIBMOD

A diferencia de M-COMP, en donde se comparaban dos modelos de comportamiento de un grupo de agentes y luego se almacenaba su diferencia, en MAUSE se almacena un único modelo. Este modelo, representado como un *trie* con sus respectivas frecuencias relativas, se almacena en la biblioteca de modelos BIBMOD. Con la finalidad de facilitar la interpretación del modelo de comportamiento, se propone representar la información del *trie* mediante una distribución de las subsecuencias de eventos. La distribución asociada al ejemplo anterior se muestra en la Figura 4.3. Como se puede observar, los 9 nodos del *trie* forman una distribución de 9 subsecuencias de eventos.

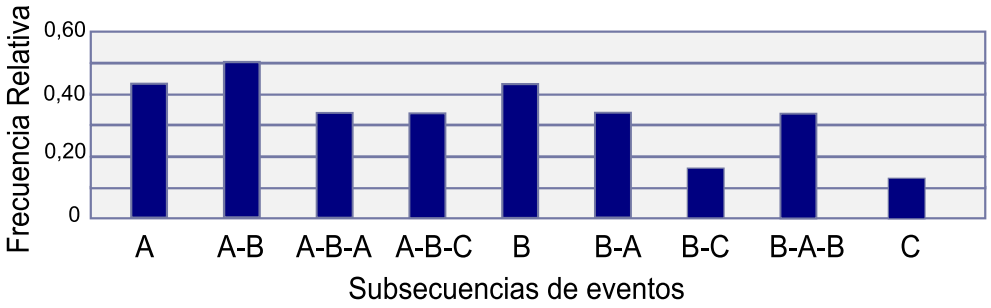


Figura 4.3: Distribución de subsecuencias de eventos considerando su frecuencia relativa.

4.2. Detección de Modelos de Comportamiento - MDMC

Una vez almacenados los modelos de comportamiento en la *Biblioteca de Modelos de Comportamiento* (BIBMOD) es posible reconocer un comportamiento observado a partir de los modelos almacenados en ésta. Dado un conjunto de modelos de comportamiento *MC* = $\{mc_1, mc_2, ..., mc_n\}$ almacenados en BIBMOD, se debe determinar qué modelo de comportamiento $mc_i \in MC$ es o se asemeja más el comportamiento observado. El algoritmo desarrollado con esta finalidad ha sido denominado ABCD (*Agent Behavior Classifier based on Distributions of events*). En los siguientes apartados se explica de forma detallada las características y el funcionamiento de ABCD.

4.2.1. Método de comparación de modelos - ABCD

Con la finalidad de aplicar test estadísticos en la comparación de modelos, ABCD utiliza la representación en forma de distribuciones de los modelos de com-

portamiento (Figura 4.3). Por otra parte, ABCD utiliza un test no paramétrico, de libre distribución, dado que no asume una distribución particular en la población. El test utilizado es una variación del test χ^2 *para dos ejemplos* y se denomina χ^2_Obs.

Para aplicar este test, el modelo que representa las observaciones se considera un ejemplo observado (distribución observada) y cada uno de los modelos almacenados en BIBMOD son considerados como los ejemplos esperados (distribución esperada). Así, la distribución observada se compara objetivamente con todas las distribuciones esperadas y se evalúa el tipo de desviación en cada caso.

El test χ^2 es un ejemplo de los denominados test de ajuste estadístico, que evalúa la bondad del ajuste de un conjunto de datos observados a una determinada distribución esperada. El valor de dicho test se obtiene aplicando la ecuación 4.1.

$$\chi^2 = \frac{(Obs - Exp)^2}{Exp} \tag{4.1}$$

Al utilizar el test χ^2, todos los valores esperados deben ser comparados con el comportamiento observado. De manera que si una subsecuencia perteneciente al comportamiento observado, no se encuentra en la distribución esperada, ésta no será tomada en cuenta. Además, la distribución esperada se crea, normalmente, a partir de gran cantidad de observaciones previas porque su número de subsecuencias suele ser mayor que el de la distribución observada. Por esta razón, este test requiere realizar gran cantidad de operaciones.

Por las restricciones arriba señaladas, se propone utilizar una variante del test χ^2 que se ha denominado χ^2_Obs en el que la forma de comparar las distribuciones se modifica tal y como se muestra en la ecuación 4.2

$$\chi^2\_Obs = \frac{(Exp - Obs)^2}{Obs} \tag{4.2}$$

El test χ^2_Obs compara todos los valores de la distribución observada con los valores correspondientes en la distribución esperada. Cabe señalar que el test no penaliza aquellas subsecuencias esperadas que no aparecen en la distribución observada. Sin embargo, esto supone un menor número de operaciones matemáticas, lo que representa una ventaja cuando el número de comportamientos almacenados es elevado.

El número de subsecuencias que contiene la distribución observada en el caso del test χ^2_Obs es siempre el mismo. Esto significa que los grados de libertad son los mismos en todas las comparaciones que se realizan con los modelos esperados. Esto implica que no requiere ningún tipo de normalización. Además, a diferencia de los métodos utilizados en M-COMP, ABCD no necesita definir ningún umbral.

El resultado del test χ^2_Obs es un valor que indica la desviación entre el modelo del agente o agentes observados y el modelo almacenado en BIBMOD con el que se ha comparado. Para poder utilizar este test como clasificador, esta comparación deberá realizarse con todos los modelos almacenados en BIBMOD. Así, el modelo que obtenga el menor valor en esta comparación, es el que más se asemeja al modelo observado.

Considerando el ejemplo del apartado 4.1.3, supóngase que dicha distribución representa el comportamiento observado de un agente. Además, considérese que BIBMOD tiene almacenados dos modelos de comportamiento diferentes obtenidos a partir de las dos secuencias: $\{A \rightarrow B \rightarrow D\}$ y $\{B \rightarrow A \rightarrow B\}$. Como se muestra en la Figura 4.4, la primera de las secuencias representa el *Modelo de comportamiento 1* y la segunda el *Modelo de comportamiento 2*. Con la finalidad de detectar qué comportamiento, de los dos almacenados, es más parecido al comportamiento observado, se aplica el test χ^2_Obs con la distribución observada y cada una de las dos distribuciones esperadas. Así, la primera comparación sería:

$\chi^2\_Obs = \frac{(0{,}33-0{,}44)^2}{0{,}44} + \frac{(0{,}5-0{,}5)^2}{0{,}5} + \frac{(0-0{,}33)^2}{0{,}33} + \frac{(0-0{,}33)^2}{0{,}33} + \frac{(0{,}33-0{,}44)^2}{0{,}44} + \frac{(0-0{,}33)^2}{0{,}33} + \frac{(0-0{,}16)^2}{0{,}16} + \frac{(0-0{,}33)^2}{0{,}33} + \frac{(0-0{,}11)^2}{0{,}11} = 1{,}645.$

Aplicando χ^2_Obs del mismo modo a la distribución del comportamiento 2, se obtiene el valor de 1,484. Como este segundo valor es menor, el comportamiento observado es más parecido al comportamiento 2.

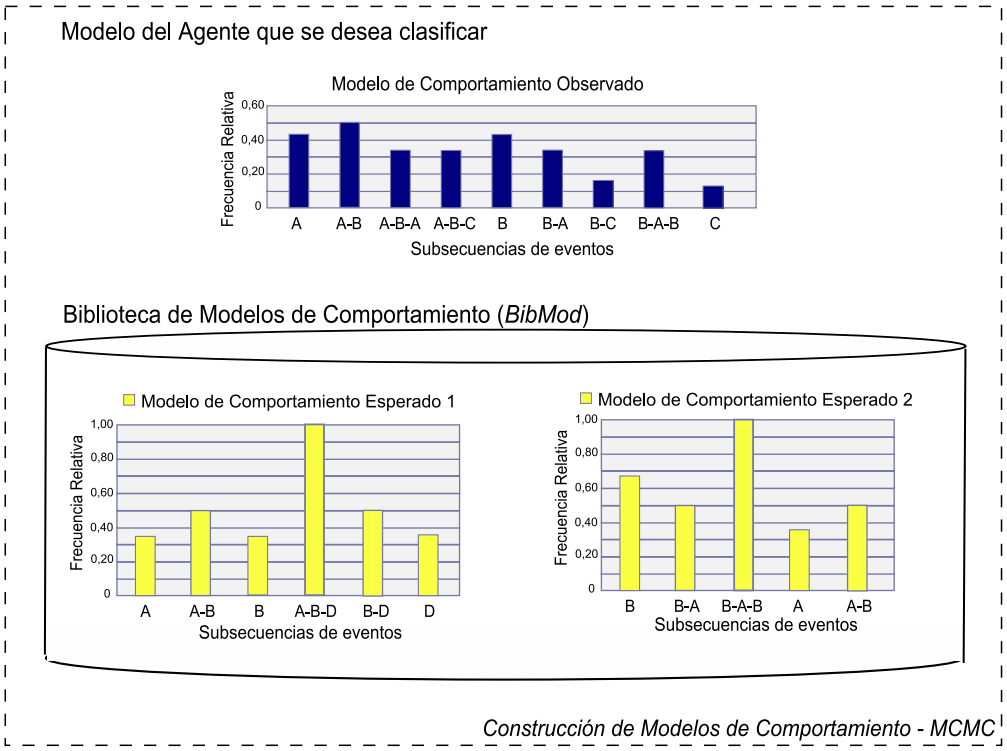
4.3. Evaluación: *RoboCup Coach*

Con la finalidad de evaluar MAUSE, se han realizado una serie de experimentos en dos dominios muy diferentes tanto en sus propiedades como en su finalidad: la detección de comportamiento de un equipo de fútbol simulado (*RoboCup Coach*) y la clasificación de usuarios de una interfaz de comandos. El primero de estos dominios fue utilizado en la evaluación de M-COMP (apartado 3.6).

Tal y como se ha descrito en el apartado 3.6 este dominio está basado en la competición *Coach 2006*. El objetivo de esta competición es modelar y detectar el comportamiento de un equipo de fútbol.

4.3.1. Aplicación de MAUSE

A la hora de aplicar MAUSE, primero han de crearse las secuencias de eventos que describen el comportamiento (MCSE). A continuación, el MCMC crea y almacena los modelos para que finalmente el MDMC detecte o clasifique un comportamiento o clasifique un comportamiento observado.



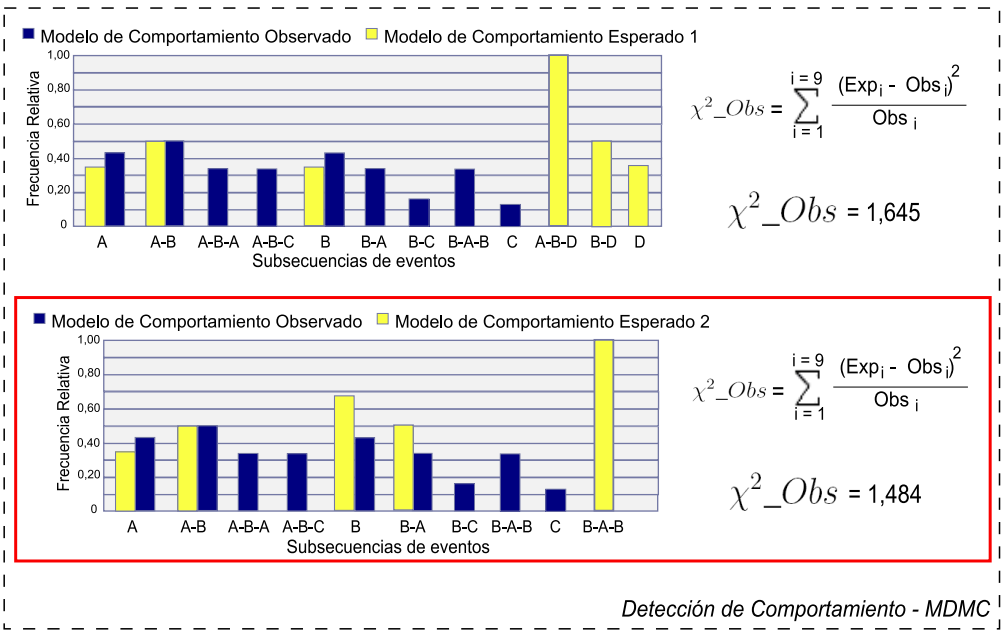


Figura 4.4: Funcionamiento de ABCD - Ejemplo

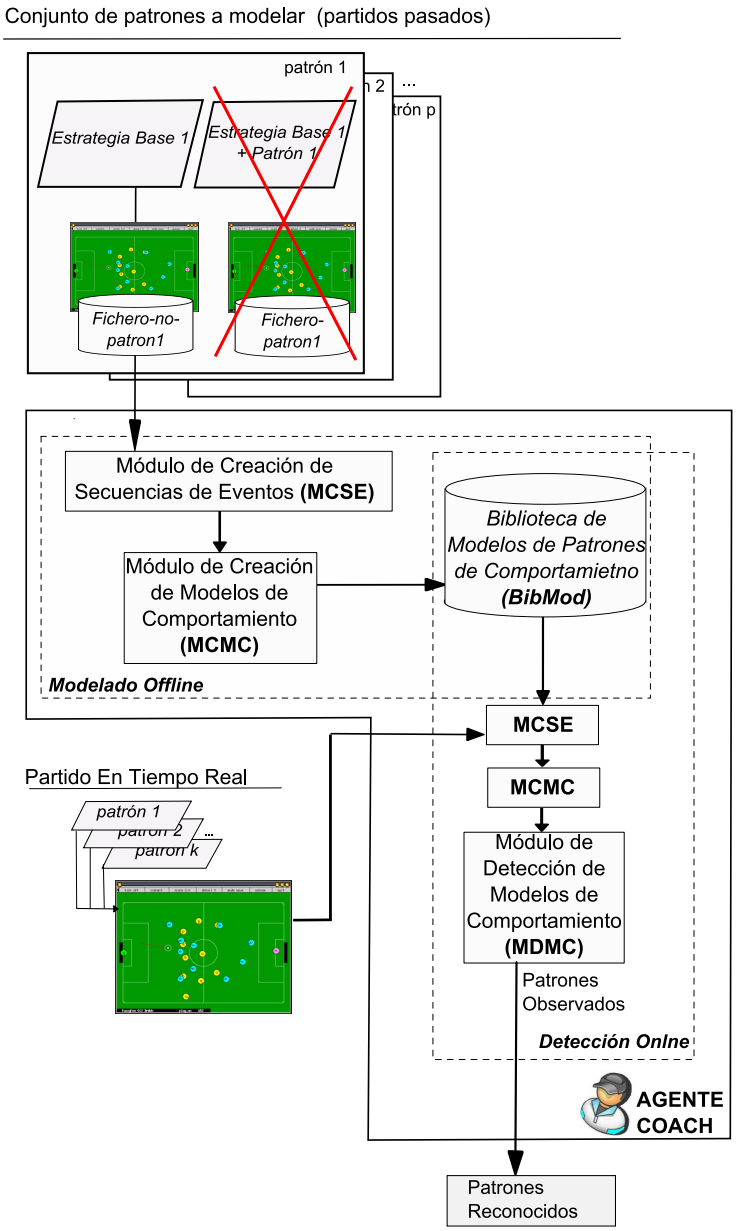


Figura 4.5: Aplicación de MAUSE en la competición *Coach*

En este dominio, el proceso de obtención de secuencias es realizado por el MCSE es el mismo que el descrito en el apartado 3.1.

En relación a la función del MCMC en este dominio, cabe señalar que a diferencia de M-COMP, MAUSE sólo utiliza las observaciones correspondientes al patrón de comportamiento que se desea modelar (Figura 4.5). En otras palabras, MAUSE no compara las observaciones del *comportamiento base* y el *comportamiento*

base + *patrón*, sino que modela sólo este último. Al igual que en M-COMP, este proceso se lleva a cabo por lotes, almacenando al final todos los comportamientos observados en BIBMOD.

Una vez creada la biblioteca de modelos durante el partido en tiempo real contra un equipo que implementa algunos de los patrones de comportamiento que se han analizado, el MDMC es capaz de reconocer dichos patrones. Partiendo de una serie de secuencias de eventos generados por el MCSE, el MDMC compara el modelo del comportamiento observado representado como una distribución, con las distribuciones de los modelos almacenados en BIBMOD. A diferencia del enfoque que utilizaba la competición (*Coach 2006*) en donde se valoraba positivamente la rapidez con que se detectaba un patrón de comportamiento, en los experimentos realizados con MAUSE, todos los patrones detectados se reportan al final del partido.

4.3.2. Configuración Experimental

La experimentación llevada a cabo en este dominio sigue las normas de la competición *Coach 2006 [184]*. Al igual que en la evaluación de M-COMP, en el primer bloque de experimentos, se analizan 17 patrones de comportamientos diferentes y a continuación, se observan 3 partidos en los que el equipo oponente implementa 4 o 5 de los patrones analizados. En el segundo bloque se analizan 16 patrones y luego se observan 3 partidos con 4 o 5 patrones activos.

4.3.3. Resultados

En la Figura 4.6 se muestran los patrones activos en cada partido en función del número de veces que se ejecutan. Mediante un rectángulo rojo se muestran los patrones detectados correctamente por MAUSE.

Cuando el MCMC compara un patrón en un modelo de comportamiento almacenado en BIBMOD utilizando el ABCD, el resultado es un valor que indica el grado de similitud entre ambos. Por esta razón, la salida de MAUSE es una lista ordenada encabezada por aquellos patrones que más se asemejan al comportamiento observado. La Tabla 4.1 muestra los resultados obtenidos al aplicar MAUSE en cada una de las iteraciones correspondientes a las dos primeras rondas de la *Coach 2006*. Los patrones activos están resaltados en negrita y marcados con un asterisco. El número de patrones activos en cada iteración está indicado entre paréntesis en el encabezado de cada iteración. Se puede apreciar que en la primera iteración de la ronda 1, MAUSE determina que el patrón que más se asemeja al comportamiento del equipo observado es *patrón_04*, y el que menos se asemeja es el *patrón_07*. Dado que en esta iteración se han activado 4 patrones, al conside-

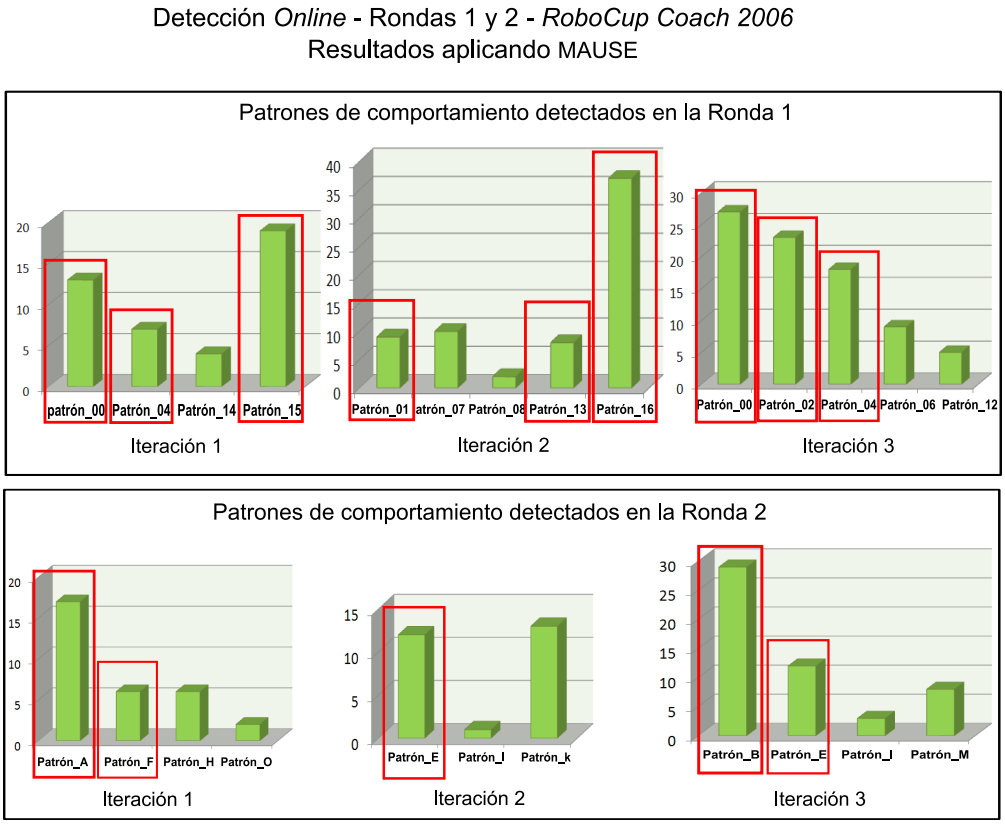


Figura 4.6: Resultados aplicando MAUSE e utilizando métodos basados en Frecuencia.

ran los 4 primeros patrones de la lista, se observa que 3 de ellos se han detectado correctamente mientras que un patrón no es detectado correctamente (*patrón_14*).

Analizando los resultados, se observa en la primera ronda que la mayoría de los patrones activos ocupan las primeras posiciones de la lista que devuelve MAUSE. En otras palabras, son detectados. Por otra parte, el número de veces que se ejecuta un patrón es determinante a la hora de detectar el patrón (Figura 4.6). En relación con los patrones que no se han detectado correctamente, generalmente están relacionados con eventos o acciones que no toma en cuenta MAUSE o se repiten muy pocas veces.

Al comparar estos resultados con los obtenidos utilizando M-COMP, se observa que los resultados de MAUSE son mejores (Tabla 4.2).

Por otra parte, a diferencia de M-COMP, MAUSE no considera las observaciones de las estrategias base del equipo oponente, por lo que el proceso de creación de modelos es más rápido. Además, la técnica estadística utilizada al comparar comportamientos en MAUSE, no necesita definir ningún umbral.

Tabla 4.1: Resultados de aplicar MAUSE en el entorno propuesto basado en la competición *Coach 2006*. Ronda 1 y Ronda 2.

Ronda 1			Ronda 2		
Iter1 (4)	*Iter2* (5)	*Iter3* (5)	*Iter1* (5)	*Iter2* (4)	*Iter3* (4)
patrón_04 *	***patrón_16 ****	***patrón_04 ****	***patrón_J ****	patrón_B	patrón_F
patrón_16	***patrón_01 ****	***patrón_02 ****	patrón_L	patrón_F	***patrón_E ****
patrón_00 *	patrón_00	patrón_13	***patrón_F ****	patrón_A	patrón_J
patrón_15 *	***patrón_13 ****	patrón_05	patrón_E	***patrón_E ****	***patrón_B ****
patrón_12	patrón_05	***patrón_00 ****	***patrón_A ****	patrón_J	patrón_D
patrón_05	***patrón_07****	***patrón_12 ****	patrón_D	***patrón_K ****	patrón_H
patrón_09	patrón_03	patrón_01	patrón_G	patrón_D	patrón_A
patrón_03	patrón_09	***patrón_06 ****	patrón_P	***patrón_P ****	patrón_P
patrón_01	patrón_10	patrón_03	patrón_C	patrón_H	***patrón_M****
patrón_06	patrón_06	patrón_07	patrón_N	patrón_C	patrón_C
patrón_08	***patrón_08****	patrón_10	patrón_M	patrón_O	patrón_K
patrón_13	patrón_15	patrón_16	patrón_K	patrón_G	patrón_O
patrón_10	patrón_02	patrón_11	patrón_B	patrón_M	patrón_G
patrón_11	patrón_12	patrón_08	***patrón_H ****	patrón_N	patrón_N
patrón_14 *	patrón_04	patrón_15	patrón_I	***patrón_I ****	***patrón_I ****
patrón_02	patrón_11	patrón_09	***patrón_O ****	patrón_L	patrón_L
patrón_07	patrón_14	patrón_14			

Tabla 4.2: Comparación de resultados aplicando M-COMP y MAUSE en el dominio de competición *Coach 2006*. Ronda 1 y Ronda 2.

		Patrones detectados por M-COMP	Patrones detectados por MAUSE
Ronda 1	Iteración 1	patrón_15, patrón_14	patrón_04, patrón_00, patrón_15
	Iteración 2	patrón_16, patrón_07	patrón_16, patrón_01, patrón_13
	Iteración 3	patrón_00, patrón_02	patrón_04, patrón_02, patrón_00
Ronda 2	Iteración 1	patrón_F, patrón_J	patrón_A, patrón_F, patrón_J
	Iteración 2	patrón_K	patrón_E
	Iteración 3	patrón_B	patrón_B, patrón_E

4.4. Evaluación: Interfaz de Línea de Comandos

Una interfaz de línea de comandos [1] es un programa informático que actúa como interfaz de usuario para comunicar al usuario con el sistema operativo utili-

[1]También denominado como intérprete de órdenes, intérprete de línea de órdenes, intérprete de comandos, terminal, consola, shell o su acrónimo en inglés CLI (por Command Line Interface)

zando únicamente órdenes escritas (texto) por el usuario en el teclado.

El objetivo de este dominio es analizar el comportamiento de un usuario a partir de los comandos que éste ha tecleado durante un determinado período de tiempo en la línea de comandos. El modelo de comportamiento del usuario, también denominado en este caso *perfil de usuario*, se deberá extraer utilizando, únicamente la secuencia de comandos, ya que no se dispone de ningún otro tipo de información acerca del usuario. Una vez creados los perfiles de usuario, se podrá, a partir de una secuencia de comandos, detectar el usuario que está interactuando con el ordenador, o determinar el tipo de usuario que es.

Dado que en este caso se trata de modelar el comportamiento de una persona, se puede considerar este dominio como un ejemplo de modelado de usuario [198]. Al obtener información del usuario mediante la observación de su comportamiento, permite que estos no interfieran en la definición de su comportamiento. Este aspecto es importante dado que los humanos normalmente realizan determinados tipos de acciones sin ser conscientes de ellos y en algunas ocasiones no pueden describir cómo se están comportando realmente [85]. Por ejemplo, a muchas de las acciones que se realizan en la conducción de un vehículo no se les presta atención por lo que sería difícil recordarlas y describirlas. Hay que tener en cuenta que la creación de perfiles usuarios es una tarea compleja debido a que los humanos suelen comportarse de forma errática y su comportamiento puede cambiar en función de las metas que deseen alcanzar.

Modelar usuarios utilizando una interfaz de línea de comandos puede ser de gran utilidad para realizar tareas diversas tales como detectar si un usuario es un intruso o predecir el comportamiento de un usuario con la finalidad de facilitarle su interacción con el sistema mostrándole información relevante o lanzando sus aplicaciones favoritas antes de que las solicite.

4.4.1. El dominio

En este dominio, la interfaz de línea de comandos que se ha utilizado es el del Sistema Operativo Unix. Este S.O. es portable, multitarea y multiusuario y fue desarrollado inicialmente por un grupo de empleados de los laboratorios *Bell* de *AT&T* [136]. Unix dispone de un intérprete de comandos[2] que permite al usuario, mediante una interfaz de línea de comandos, introducir instrucciones al sistema.

Unix shell es el nombre que se le da al intérprete de comandos en los sistemas de la familia Unix. También existen otros intérpretes de comandos como por ejemplo, el *command.com* que pertenece a los sistemas *MS-DOS*, *Windows 95* y *98*, y *cmd.exe* que es el equivalente en los sistemas *Windows* de la familia *NT* y *XP*.

[2]también denominado *shell*.

4.4.2. Conjunto de datos

En este dominio se posee un número elevado de comandos asociado a distintos usuarios. Los comandos provienen de ficheros históricos y se analizan para obtener una secuencia de elementos (comandos).

MAUSE ha sido evaluado utilizado dos fuentes de datos:

- **Conjunto de 9 Usuarios**: Conjunto de datos obtenidos de ficheros históricos del intérprete de comandos de Unix de 9 usuarios durante 2 años en la Universidad de Purdue [123]. Cada fichero contiene entre 10.000 y 60.000 comandos tecleados por un usuario. Estos datos están disponibles en la página web del repositorio UCI de aprendizaje automático [125].

- **Conjunto de 50 Usuarios**: Este conjunto de datos está formado por los comandos tecleados por 50 usuarios Unix durante un período de tiempo largo. Estos datos han sido utilizados por Schonlau et al. [163] en investigaciones sobre la detección de intrusos. Schonlau intercala comandos de otros usuarios en el conjunto de datos original como si hubieran sido tecleados por intrusos. Sin embargo, en esta evaluación, se utilizan los datos originales de los 50 usuarios. Existe un fichero de 15.000 comandos por cada usuario. Estos datos están disponibles en la página web de Matt Schonlau [132].

En los dos conjuntos de datos se han insertado unas palabras especiales que indican el inicio y fin de sesión. Estas palabras especiales forman parte del conjunto de datos y son consideradas como comandos en la secuencia asociada. A pesar de que en estos conjuntos de datos también se dispone de información adicional como por ejemplo, cuánto tiempo se tarda en teclear un determinado comando, en estos experimentos se utiliza únicamente la secuencia de comandos tecleada por un usuario.

4.4.3. Aplicación de MAUSE

Al aplicar MAUSE, lo primero que se realiza es obtener la secuencia de eventos que representa el comportamiento de un usuario utilizando el MCSE. En este caso, cada evento está representado por un comando del intérprete de comandos de Unix.

Como se extrae de la experimentación realizada en [122], en este dominio, el número de *palabras* diferentes resultante de todas las secuencias es muy amplio, pero la frecuencia de muchos comandos es muy baja. Por esta razón y para no identificar a los usuarios en función de los nombres de ficheros que éstos utilizan, se aplica técnica presentada en [122], en donde se omiten los nombres de ficheros y se sustituyen por un número que representa la cantidad de nombres de

ficheros que aparecen de forma contigua. Por ejemplo, si los comandos son: `cat fichero1.txt fichero2.txt fichero3.txt`, estas cuatro *palabras* serán sustituidas por: `cat` ⟨3⟩, donde ⟨3⟩ indica que se añadieron tres ficheros. Los nombres de usuarios y las estructuras de directorios también se procesan del mismo modo. De esta manera, las dos siguientes sesiones:

```
Start session 1
cd docs
ls -laF
cat a.txt b.txt c.txt
exit
End session 1
```

```
Start session 2
cd games
xquake
fg
vi scores.txt
mailx jhon@bb.es
exit
End session 2
```

son representadas por las siguientes dos secuencias de comandos:

```
**SOF**
cd
⟨1⟩
ls
-laF
cat
⟨3⟩
exit
**EOF**
```

```
**SOF**
cd
⟨1⟩
xquake
fg
vi
⟨1⟩
mailx
⟨1⟩
exit
**EOF**
```

Obtenidas las secuencias de comandos, se crean modelos (perfiles) de usuarios a partir de dichas secuencias aplicando el MCMC. Creados estos perfiles y almacenados en BIBMOD, se debe reconocer el comportamiento de un usuario utilizando el MDMC.

Para poder evaluar MAUSE, el conjunto de comandos de un usuario se divide de forma que con una parte de ellos se crea el modelo del usuario (*secuencia de entrenamiento*), y con el resto se evalúa la clasificación realizada por el MDMC (*secuencia de evaluación*). Esta idea queda representada en la Figura 4.7.

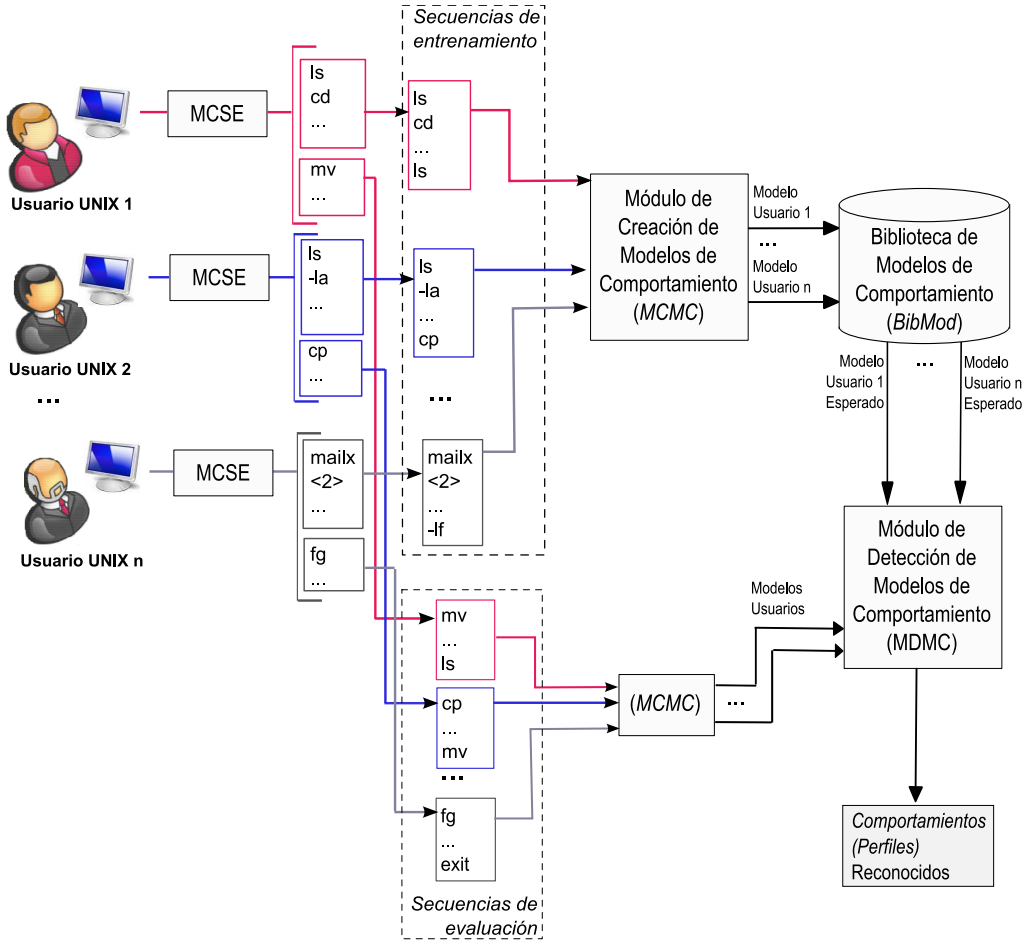


Figura 4.7: Comandos Unix: Aplicación de MAUSE

4.4.4. Configuración Experimental

Para poder evaluar MAUSE considerando todo el conjunto de datos, se utiliza *validación cruzada*. Para esta experimentación y debido a la gran cantidad de datos de los que se dispone, se utiliza validación cruzada con 10 particiones. Para ello, los comandos tecleados por un único usuario se dividen en 10 subconjuntos disjuntos del mismo tamaño. El modelo de un usuario se crea con 9 de los subconjuntos y el subconjunto de cada uno de los usuarios que no se ha utilizado, se utiliza para validar el clasificador ABCD. Este proceso se repite 10 veces.

El número de comandos que se utiliza para realizar la evaluación influye en los resultados de la clasificación. Por este motivo, se han realizado una serie de experimentos con diferentes tamaños de secuencias de comandos. Se han utilizado secuencias de 50, 100, 500, 1.000, 5.000 y 10.000 comandos para entrenamiento y evaluación con la finalidad de determinar la relevancia de este tamaño en el resul-

tado. Esta selección se ha realizado considerando los últimos comandos tecleados por cada usuario.

Además, en el MCMC, la *longitud de las subsecuencias* en las que se divide la secuencia de comandos, es un parámetro muy relevante. Utilizando subsecuencias más largas, el tiempo que se consume al crear el *trie* es mayor y el número de subsecuencias de la distribución se incrementa considerablemente. Por esta razón, se han utilizado tres valores diferentes para segmentar la secuencia de comandos; tres, cinco y diez.

4.4.5. Resultados

MAUSE es capaz de generar una lista ordenada de usuarios cuyos perfiles se asemejan al comportamiento observado, tal y como se hizo en el dominio de la *RoboCup Coach*. Además, esta lista almacena el valor obtenido al aplicar el test χ^2_Obs con cada caso. Sin embargo, en estos experimentos sólo se considera correcta la clasificación si el perfil que ocupa la primera posición de la lista es el del usuario que tecleó la secuencia de comandos observada.

Los resultados obtenidos en este dominio se muestran en la Tabla 4.3. La desviación estándar mide la dispersión de todos los valores del test χ^2_Obs almacenados en la lista generada. Dado que se utilizó validación cruzada con 10 conjuntos, los resultados para secuencias de 50 comandos utilizando subsecuencias de longitud 10 para crear el *trie*, no pueden ser calculados porque las secuencias de evaluación tienen únicamente 5 comandos, el 10 % de los 50 comandos.

Analizando los resultados del *conjunto de 9 Usuarios*, mostrados en las columnas centrales de la tabla, se observa que la tasa de aciertos es elevada (entre 76,67 % y 91,11 %). Incluso utilizando solamente 50 comandos (*secuencias de entrenamiento* de 45 comandos y de *evaluación* de 5 comandos), la tasa de aciertos es cercana al 80 %. Por otra parte, los resultados no varían mucho en función de la longitud de la subsecuencia utilizada para crear el *trie* (3, 5 y 10), tal y como se podría haber pensado *a priori*. Esto se debe a que es poco frecuente que un usuario repita con exactitud una secuencia de 10 comandos. Por esta razón, si se crea el perfil de un usario utilizando subsecuencias de 10 elementos, es probable que se esté incorporando algo de ruido en los datos lo que provoca el decremento de la tasa de aciertos. Los mejores resultados se obtienen utilizando subsecuencias de longitudes 3 y 5. Por lo tanto, habrá varios patrones de esta longitud en una misma sesión, lo cuál define a un determinado usuario y MAUSE es capaz de reconocerle.

Los resultados del *conjunto de 50 Usuarios* tienen una tasa de acierto menor debido al número de perfiles que MAUSE debe considerar. En este caso, esta tasa se incrementa considerablemente al incrementar el número de comandos. Utilizando 10.000 comandos (9.000 para *entrenamiento* y 1.000 para *evaluación*) la

tasa de aciertos está cercana al 80 %, lo cual es un buen resultado si se considera el número elevado de usuarios.

Tabla 4.3: Porcentaje de aciertos en la clasificación utilizando MAUSE. Validación cruzada con 10 particiones. Conjuntos de 9 y 50 Usuarios.

		Resultados Clasificación utilizando MAUSE			
		Conjunto de 9 Usuarios		**Conjunto de 50 Usuarios**	
Número de Comandos	Longitud Subsec	Tasa de Aciertos (%)	Desviación Estándar	Tasa de Aciertos (%)	Desviación Estándar
50	3	80,00	1,40	48,20	8,99
	5	78,89	1,34	48,80	7,73
100	3	80,00	0,96	53,40	8,42
	5	76,67	1,08	51,40	9,81
	10	78,89	0,83	54,80	6,99
500	3	90,00	1,08	64,00	9,16
	5	91,11	1,27	64,20	10,17
	10	86,67	1,49	63,80	12,48
1.000	3	87,78	1,53	72,00	10,14
	5	87,78	1,30	71,20	10,49
	10	81,11	1,84	69,00	11,69
5.000	3	85,56	1,23	75,80	12,05
	5	87,78	1,30	76,60	12,26
	10	84,40	1,54	75,00	12,64
10.000	3	88,87	1,53	76,20	12,14
	5	90,00	1,40	78,80	13,14
	10	87,67	1,66	76,60	13,39

4.4.6. Comparación con *Modelos Ocultos de Markov*

Los resultados obtenidos por MAUSE en este dominio, se comparan con los que obtiene un clasificador basado en los *Modelos Ocultos de Markov* (*Hidden Markov Models*, HMM) [2]. El Apéndice B muestra una introducción a estos modelos estadísticos. Los HMM se han seleccionado como base para esta comparación debido a su demostrada utilidad en dominios basados en secuencias de datos [24] en áreas como bioinformática [16, 169] o la robótica [87, 161]. Además, ciertas investigaciones muestran su buen funcionamiento en el modelado de usuarios [124, 123] y reconocimiento de actividades [29, 62]. Esto se debe a que no sólo se basan en una rigurosa teoría matemática, sino que los modelos se ge-

neran mediante un proceso de aprendizaje sobre conjuntos de datos reales, lo cual garantiza su funcionalidad.

Para aplicar HMM al dominio, es necesario determinar la estructura de cada modelo; es decir, el número de estados y la topología. En este contexto, la topología está determinada por las conexiones entre los distintos estados que forman el HMM. A continuación, se detalla el proceso de construcción de un clasificador basado en HMM para este dominio:

1. *Creación de los ficheros de entrenamiento:* cada usuario tendrá asociado un fichero con su correspondiente *secuencia de comandos de entrenamiento*. Es decir, una parte de los comandos que un usuario ha tecleado.

2. *Creación y entrenamiento del sistema:* cada usuario tiene un HMM asociado que representa su modelo de comportamiento. Para crear el HMM asociado a un usuario a partir de su fichero de entrenamiento son necesarios dos valores:

   - Longitud del alfabeto, M: se debe calcular el número de comandos diferentes que cada usuario ha tecleado.
   - Número de estados ocultos, N: La elección del valor N afecta directamente a cómo describe el HMM un determinado comportamiento por lo que una buena elección es fundamental [2]. Cuando los estados ocultos tienen una interpretación clara del dominio, este valor puede ser impuesto por el dominio. Sin embargo, si no está disponible, deberá realizarse una búsqueda empírica para obtener el valor más apropiado. En este caso, y dado que este sistema se compara con MAUSE, el número de estados ocultos se corresponde con la *longitud de subsecuencias* debido a que que ambos valores miden el número de comandos dependientes en una secuencia.

   A partir de los valores M y N, se entrenará cada uno de los HMM que representan el modelo de comportamiento de cada usuario; obteniendo así la matriz de probabilidades de transición de estados (A) y la probabilidad de distribución en cada estado (B). Dado que la longitud del alfabeto suele ser un número muy elevado, la dimensión de las matrices creadas por cada usuario es muy alta.

3. *Clasificación*: En este caso, clasificar una determinada secuencia de comandos como uno de los usuarios modelados consiste en calcular la probabilidad de que dicha secuencia pertenezca a cada uno de los modelos (HMM) creados. Para el cálculo de esta probabilidad se utiliza el *Algoritmo Forward* que se explica en el Apartado B.3 del Apéndice B. Mediante este algoritmo

se obtiene la probabilidad de que la secuencia observada pertenezca a cada uno de los HMM. La secuencia será clasificada como aquél usuario (representado por un HMM) cuya probabilidad sea más alta.

Utilizando los mismos conjuntos de datos que en la evaluación de MAUSE, los modelos de comportamiento de los usuarios basados en HMM y su posterior clasificación se han creado utilizando la herramienta Umdhmm [106].

La Tabla 4.4 muestra los resultados obtenidos utilizando validación cruzada con 10 particiones. Como se ha indicado, el número de estados está relacionado con la longitud de la subsecuencia de elementos utilizada en MAUSE, lo que permite comparar ambos métodos. Los resultados muestran que partiendo de una secuencia de 50 o 100 comandos, un clasificador basado en HMM obtiene una tasa de aciertos baja. Para obtener resultados similares a los alcanzados con MAUSE, el número de comandos utilizados para el entrenamiento debe ser alto. Aun así, la diferencia entre la tasa de aciertos de HMM y MAUSE en el *conjunto de 50 usuarios* es alta. Esta tasa de aciertos es levemente mayor utilizando HMM con una gran cantidad de datos de entrenamiento (10.000 comandos).

El número de estados ocultos utilizados para crear el HMM es también relevante en este dominio, pero tal y como ocurría con las longitudes de subsecuencias en MAUSE, su valor óptimo debe ser calculado de forma experimental. A pesar de que un número mayor de estados hace más complejo el HMM, esto no implica que la tasa de aciertos se incremente. Al igual que ocurría en MAUSE, en la mayoría de los resultados la tasa de aciertos más elevada se obtiene con 5 estados ocultos. La utilización de 10 estados no sólo incrementa mucho la información que se tiene que almacenar sino que introduce ruido.

En las Figuras 4.8 y 4.9 se comparan los resultados de MAUSE y de los HMM. Cabe destacar la alta tasa de aciertos de MAUSE sobre secuencias con pocos comandos, lo cuál indica lo rápido que este enfoque es capaz de crear buenos perfiles de usuario.

4.5. Conclusiones

En este capítulo se ha presentado MAUSE, un enfoque capaz de modelar y detectar comportamientos de agentes en entornos multiagente. La aplicación de este enfoque implica representar un comportamiento como una secuencia de eventos. MAUSE utiliza métodos basados en frecuencia por sus buenos resultados en la minería de secuencias de datos [3], pero podría ser modificado utilizando así otros métodos diferentes como los basados en dependencia estadística utilizados en M-COMP. MAUSE ha sido evaluado en dos dominios muy diferentes: la competición

Tabla 4.4: Porcentaje de aciertos en la clasificación utilizando HMM. Validación cruzada con 10 particiones. Conjuntos de 9 y 50 Usuarios.

		Resultados Clasificación utilizando HMM			
		Conjunto de 9 Usuarios		**Conjunto de 50 Usuarios**	
Nº de Comandos	Nº de Estados	Tasa de Aciertos (%)	Desviación Estándar	Tasa de Aciertos (%)	Desviación Estándar
50	3	52,22	2,23	30,40	14,08
	5	54,44	2,06	32,40	14,72
	10	54,44	2,08	34,80	15,02
100	3	64,44	1,49	39,40	8,72
	5	61,11	1,53	40,00	8,58
	10	62,22	1,60	40,40	8,94
500	3	63,33	1,22	42,20	6,19
	5	68,89	1,30	48,20	6,03
	10	66,67	1,26	50,00	5,86
1.000	3	63,33	1,20	46,20	4,69
	5	68,89	1,32	49,20	4,55
	10	66,67	1,09	49,80	4,47
5.000	3	80,00	1,05	54,20	3,89
	5	82,22	0,90	58,20	3,53
	10	80,20	0,97	58,00	3,45
10.000	3	89,90	1,22	76,40	3,35
	5	93,32	1,32	78,80	3,41
	10	91,00	0,97	77,80	3,29

Coach y la clasificación de usuarios del intérprete de comandos Unix. Esto demuestra lo general del enfoque propuesto.

Respecto al entorno *RoboCup Coach*, y al igual que ocurría con el enfoque planteado en el capítulo anterior, hay que resaltar la complejidad que requiere detectar comportamientos en un entorno multiagente. Esto se debe a que en el comportamiento de un agente pueden involucrarse otros agentes que modifiquen su comportamiento. A pesar de esto, los resultados obtenidos aplicando MAUSE muestran su capacidad para detectar y clasificar un número importante de patrones de comportamiento.

Por otra parte, los experimentos en la clasificación de usuarios de la interfaz de comandos de Unix, muestran buenos resultados en este dominio. La comparación con los *Modelos Ocultos de Markov*, muestra que MAUSE es capaz de modelar bien el perfil de comportamiento de un usuario incluso a partir de pocos coman-

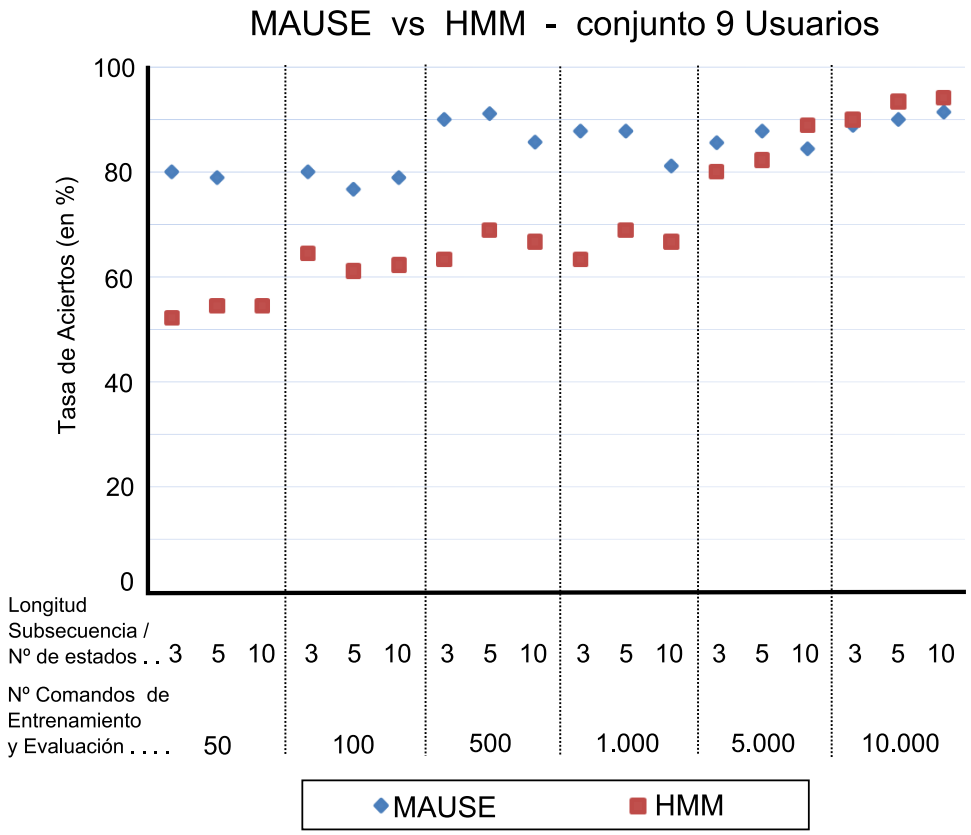


Figura 4.8: MAUSE vs HMM - Porcentaje de aciertos en la clasificación. Validación cruzada con 10 particiones. Conjuntos de 9 Usuarios.

dos. A diferencia de lo que se podría suponer *a priori*, una mayor longitud de subsecuencia utilizada, no implica la creación de mejores perfiles. Esto se debe a que es poco frecuente que un usuario repita de forma exacta una secuencia de comandos alta. Si el perfil de un usuario se crea con subsecuencias de muchos comandos, es probable que se introduzca ruido en los datos.

Por último, se debe considerar que en ambos dominios se dispone de información adicional en cada uno de los eventos que se tratan. Por ejemplo, cuánto tiempo se tarda en ejecutar una determinada acción o teclear un determinado comando. En este enfoque dicha información no ha sido incluida, pero se propone como trabajo futuro la modificación de MAUSE con la finalidad de incorporar esta información.

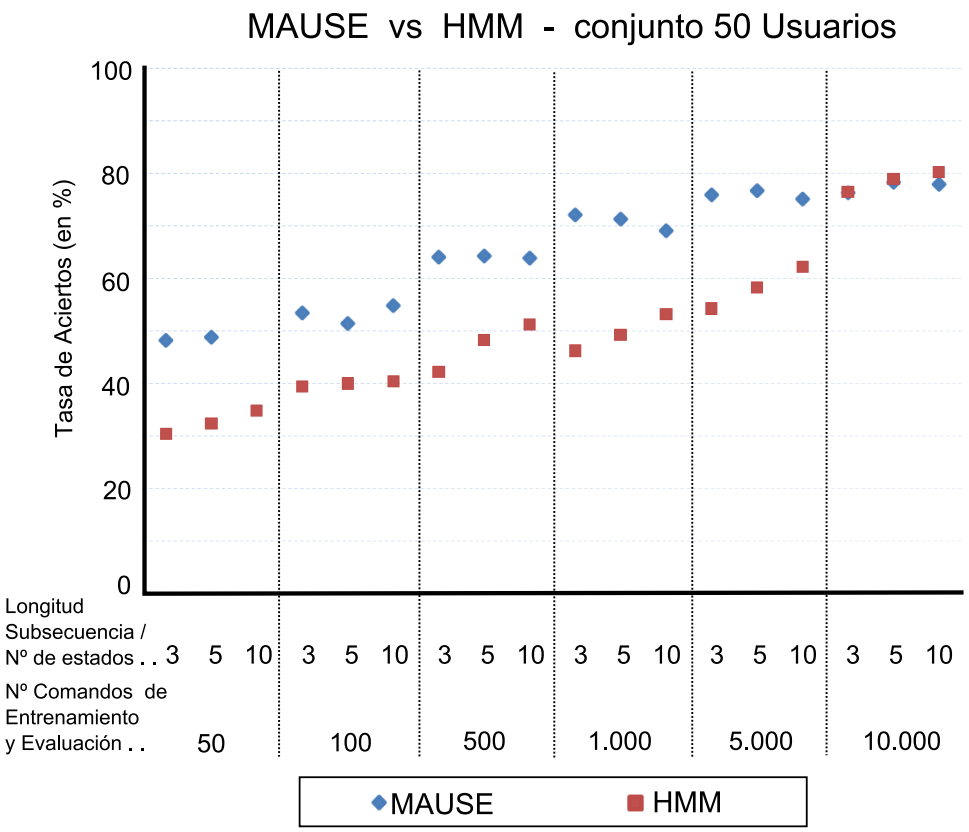


Figura 4.9: MAUSE vs HMM - Porcentaje de aciertos en la clasificación. Validación cruzada con 10 particiones. Conjuntos de 50 Usuarios.

Capítulo 5

Modelado Auto-Adaptativo de Agentes con Flujo Continuo de Eventos: M-ADAP

En este capítulo se detalla el tercer enfoque propuesto en esta tesis doctoral. Este enfoque propone el modelado de agentes mediante el flujo continuo de datos. En este caso, las observaciones de los agentes serán utilizadas para clasificar su comportamiento, así como para crear nuevos modelos o actualizar los modelos creados. Este nuevo enfoque está motivado por la necesidad de adaptar los modelos creados a las variaciones en el comportamiento que sufren los agentes en determinados entornos. Actualmente, la mayoría de los modelos de agentes necesitan ser re-diseñados o re-entrenados para adaptarse a nuevas circunstancias.

La base de este enfoque consiste en la utilización de los llamados Sistemas Auto-Adaptativos (SAA) [15], por lo que a este nuevo enfoque se ha denominado M-ADAP (*Modelado de Agentes Auto-Adaptativo*). A diferencia de M-COMP y MAUSE, este enfoque agrupa aquellos comportamientos que son semejantes, considerando así a un agente dentro de un comportamiento general y no particular.

El esquema general del funcionamiento de M-ADAP se muestra en la Figura 5.1. Las observaciones del comportamiento de uno o varios agentes se realizan en tiempo real. A partir de estas observaciones, el MCSE obtiene las secuencias de eventos que utilizará el MCMC para crear el correspondiente modelo de comportamiento. A continuación, se utiliza un nuevo módulo que no estuvo presente en los enfoques anteriores, el MGMA (*Módulo de Gestión de Modelos de comportamiento Auto-Adaptativos*). Este módulo no sólo clasifica el modelo creado, sino que utiliza dicho modelo para actualizar la biblioteca de modelos denominada BIBADAP (*Biblioteca de Modelos Auto-Adaptativos de comportamiento*). El MCSE y el MCMC se utilizan del mismo modo que en los enfoques anteriores. Al igual que en MAUSE, el MCMC utiliza la frecuencia relativa para obtener las subsecuen-

cias relevantes. En el siguiente apartado se detalla el MGMA y la estructura de la biblioteca utilizada.

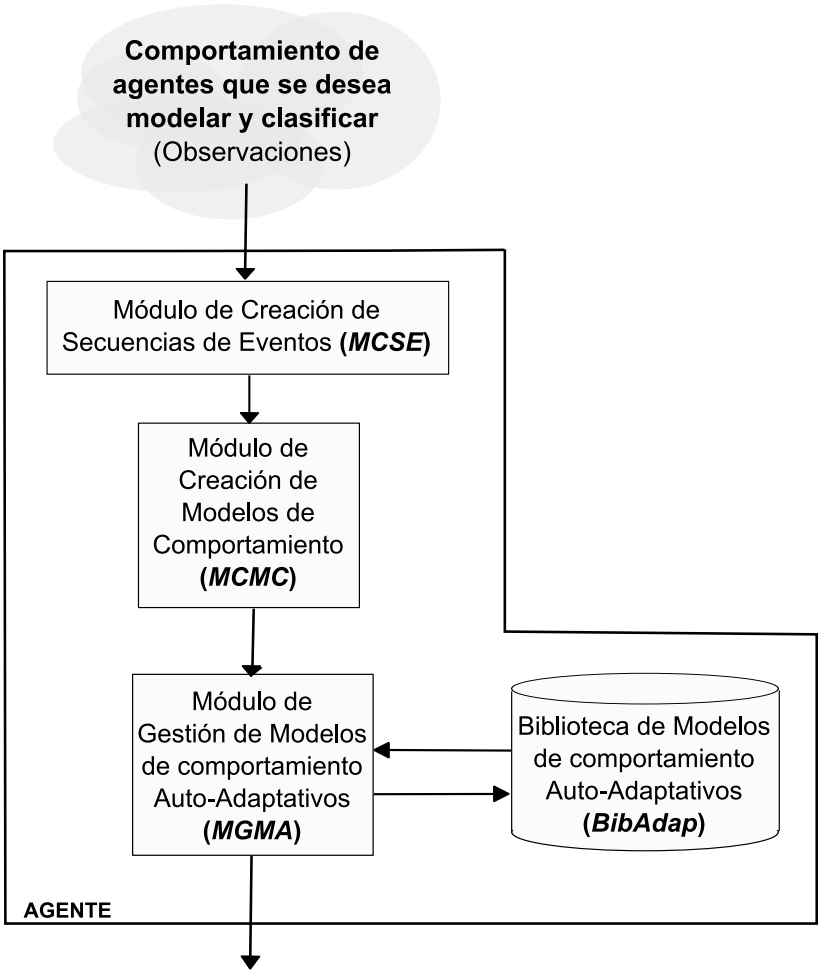


Figura 5.1: Estructura general de M-ADAP: Modelado de Agentes Auto-Adaptativo con Flujo Continuo de Eventos.

5.1. Gestión de Modelos de Comportamiento Auto-Adaptativos (MGMA)

Al igual que los enfoques anteriores, M-ADAP utiliza una biblioteca en la que se almacenan los modelos de comportamiento. Sin embargo, en este caso, un modelo representa un grupo de agentes cuyo comportamiento es similar. Este tipo de modelos es denominado prototipo.

El MGMA tiene como finalidad la actualización y utilización de los prototipos almacenados en BIBADAP. Inicialmente, la biblioteca está vacía. El primer modelo insertado, considerado como el primer prototipo, es el modelo de comportamiento del primero de los agentes observados. La actualización de esta biblioteca se realiza en función de las nuevas observaciones.

Cuando el MGMA recibe un nuevo modelo de comportamiento, éste se clasifica como de la clase correspondiente al prototipo que más se asemeja. A continuación, la biblioteca se actualiza con la finalidad de que ésta se adapte al nuevo modelo observado. Para ello, se crean, eliminan y/o modifican los prototipos almacenados en BIBADAP.

5.1.1. Representación de los modelos en el Espacio de Características

Los modelos de comportamiento utilizados como entrada del MGMA son representados como distribuciones de subsecuencias de eventos. En este enfoque, a diferencia de MAUSE, se crean todas las distribuciones a partir de los *tries*. Así, un modelo de comportamiento no es tratado como un *trie*, sino como un vector de valores en el que cada elemento representa una subsecuencia determinada y tiene asociado el valor de su frecuencia relativa. Estas distribuciones, para poder ser utilizadas por los SAA, deben representarse en un espacio de características (EC) determinado. Este EC representa un espacio n-dimensional abstracto donde cada distribución (ejemplo/modelo) se representa como un punto. La dimensión de dicho espacio se determina en función del número de características que definen las distribuciones.

Cada distribución se representa como un vector de datos que determina un punto concreto en el EC. Por lo tanto, se deben conocer *a-priori* todas las subsecuencias de eventos diferentes existentes. Sin embargo, este número de subsecuencias podría ser desconocido y además, la creación de este gran EC desde un principio resulta poco eficiente. Por este motivo, las dimensiones del EC en el que se representan las distribuciones se incrementan en función de las diferentes subsecuencias de dichas distribuciones.

Para aclarar los conceptos explicados anteriormente, considérese el dominio de línea de comando de Unix explicado en el capítulo anterior y en el que el comportamiento de un usuario se representa por subsecuencias de comandos. Si se desea representar cualquier distribución (de subsecuencias de comandos) en un EC, éste debería tener tantas dimensiones como subsecuencias diferentes existen. Sin embargo, un EC de tal magnitud resulta inabordable e imposible de crear porque *a priori* no se conocen todas las subsecuencias existentes. Para resolver este problema, el EC utilizado se va incrementando en función de las nuevas subse-

cuencias de comandos presentes en los modelos.

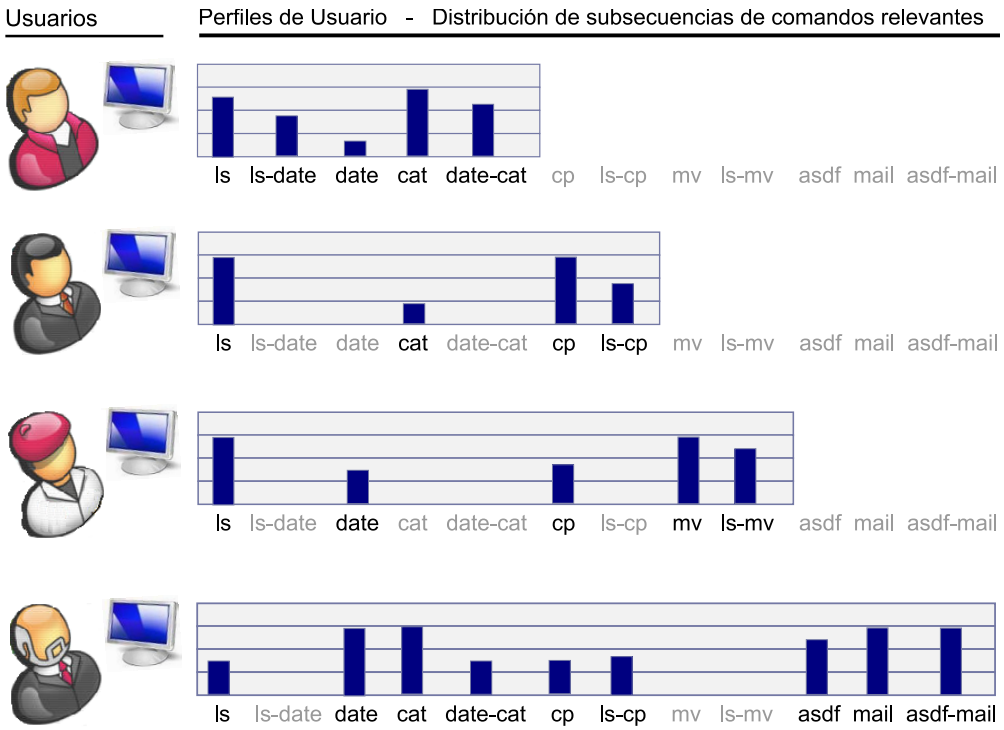


Figura 5.2: Ejemplo para la representación de las distribuciones de subsecuencias de eventos en SAA

Considerando este ejemplo, en la Figura 5.2 se muestra cómo la distribución del primer usuario está formada por cinco subsecuencias de comandos (*ls, ls-date, date, cat* y *date-cat*) por lo que para representar este ejemplo en el EC correspondiente son necesarias cinco dimensiones. Si se observa el segundo de los usuarios, se comprueba que hay subsecuencias del usuario anterior para las que este usuario no tienen valor (*ls-date, date* y *date-cat*), lo cual significa que este usuario no ha tecleado dichas subsecuencias. Sin embargo, se deben añadir dos nuevas subsecuencias (*cp* y *ls-cp*) por lo que para representar esta distribución en un EC, se debe ampliar el EC anterior. De esta manera, el EC en el que se han representados los dos usuarios posee siete dimensiones, aunque existen dimensiones para las cuáles las distribuciones no tienen valor. Es importante considerar que no tener valor no significa que éste sea cero, sino que su valor no está representado. Si se consideran los dos últimos usuarios del ejemplo, se observa que las dimensiones necesarias para representar todas las subsecuencias diferentes asciende a 12. Por lo tanto, las dimensiones del EC se corresponden con el número de subsecuencias de eventos diferentes que se quieren representar, y estas dimensiones podrán incrementarse en función de nuevas subsecuencias.

5.1.2. Algoritmo ABCD-ADAP

Este apartado detalla la estructura del algoritmo utilizado en el MGMA tanto para actualizar los prototipos almacenados en BIBADAP como para clasificar un comportamiento observado. Este algoritmo se ha denominado ABCD-ADAP (*Agent Behavior Classifier based on Distributions of events - Adaptative*).

Una importante característica de este algoritmo de clasificación es que es incremental, es decir, no necesita disponer de todos los datos *a priori*, sino que trata los ejemplos a medida que los recibe. Este tipo de algoritmos pueden ser aplicados en un gran número de entornos dinámicos donde los datos van siendo adquiridos a lo largo del tiempo, lo que obliga a procesarlos secuencialmente en episodios sucesivos, es decir, de forma incremental. Un problema de este tipo de algoritmos es que la clasificación podría depender del contexto, el cual no es recogido mediante los atributos de los datos de entrada. Si esto ocurriera, se podrían producir cambios en la clasificación por una causa oculta y el problema se conoce como *hidden context*. En este caso, un cambio en el contexto podría inducir grandes y repentinos cambios en la clasificación, produciendo lo que se conoce como *"drift"* [188, 111]. Con la finalidad de poder adaptarse a estos cambios lo más rápido posible, el algoritmo que se propone es capaz de modificar tanto sus parámetros como su estructura.

ABCD-ADAP recibe como entrada los modelos de comportamiento de uno o varios agentes generados por el MCMC. Cada modelo se representa por una distribución de secuencias de eventos y se considera como un nuevo ejemplo. Estos ejemplos se representan en un determinado EC. En este caso, se considera un EC por cada una de las clases existentes, y todos los EC tienen las mismas dimensiones. De esta manera, cada uno de los ejemplos obtenidos es representado en su correspondiente EC. En función de la homogeneidad de los ejemplos representados en un EC, éste puede tener uno o varios prototipos que representan diferentes grupos de una misma clase.

Una de las principales características que posee este algoritmo es que, a pesar de considerar todos los ejemplos para actualizar los prototipos almacenados en BIBADAP, no es necesario almacenar todos ellos. Si un ejemplo no es un prototipo, éste no se almacena, sino que únicamente se utiliza para actualizar determinados valores en la propia biblioteca. Así, BIBADAP está formada por los prototipos creados y una información muy reducida sobre todos los ejemplos recibidos. Esta información permite tener en cuenta todos los ejemplos en la actualización de los prototipos pero sin necesidad de almacenarlos en la biblioteca.

Los SAA permiten adaptar su funcionamiento de acuerdo con el tipo de tarea que se quiere realizar. Para la tarea de modelar el comportamiento de un agente o grupo de agentes, el algoritmo incremental ABCD-ADAP sigue los siguientes pasos que se repiten con cada ejemplo que recibe:

1. Clasificación de un nuevo ejemplo en una determinada clase (representada por un prototipo).
2. Cálculo del potencial del nuevo ejemplo. Esta medida determina la proximidad espacial entre el nuevo ejemplo y resto de ejemplos representados en el EC.
3. Actualización de los prototipos en función del nuevo ejemplo obtenido.
4. Inserción del nuevo ejemplo como prototipo si fuera necesario.
5. Eliminación de alguno de los prototipos insertados si fuera necesario.

A continuación se detallan cada uno de estos pasos dejando para el final el proceso de clasificación de un nuevo ejemplo.

Cálculo del potencial del nuevo ejemplo

Los SAA se basan en el concepto de potencial para definir la proximidad espacial entre un ejemplo y el resto de ejemplos representados en un determinado EC. De esta manera, se realiza la agrupación de ejemplos basándose en su potencial. Pueden utilizarse diferentes funciones para calcular el potencial, pero siempre deben ser monótonas e inversamente proporcionales a la distancia entre todos los ejemplos. La distancia entre dos ejemplos puede calcularse utilizando distancia coseno, euclídea, mahalonobis, etcétera. Cabe destacar que el valor de potencial de un ejemplo es mayor si éste está rodeado de un número elevado de ejemplos.

En los SAA, el potencial (P) de un ejemplo determinado ($vectorx_k$) en el EC de su clase, se calcula a partir de las distancias acumuladas entre dicho ejemplo y todos los ejemplos representados en dicho EC. Este valor mide la densidad de datos que rodean al ejemplo y se calcula mediante la siguiente fórmula (ecuación 5.1).

$$P(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1} distancia(x_k, x_i)}{k-1}} \tag{5.1}$$

donde x_k denota el ejemplo k-ésimo representado en el EC y *distancia* representa la cercanía entre dos ejemplos en dicho espacio. *k*-1 es el número de ejemplos representados en el EC.

La *distancia* depende del método utilizado para calcular la semejanza entre un punto y otro en el EC. En el clasificador basado en SAA propuesto en [14], el potencial se calcula utilizando la distancia euclídea, mientras que en [13] se utiliza la distancia coseno. En ABCD-ADAP, la medida utilizada para calcular la

distancia entre dos ejemplos es la distancia coseno [61]. Esta distancia se obtiene a partir del ángulo que forman dos vectores y en este caso, permite solucionar los dos problemas que se detallan a continuación:

1. Es posible que se necesite calcular la distancia entre ejemplos (distribuciones) con un número de elementos distinto. Los ejemplos (distribuciones) representados en un EC pueden tener un número distinto de elementos. En otras palabras, se debe poder comparar vectores con distinto número de elementos (distribuciones con distinto número de subsecuencias).

2. Dado que es posible que todos los ejemplos no tengan el mismo número de elementos, algunos ejemplos no tendrán valor para ciertos elementos. En este caso, el que un elemento no tenga valor, no significa que éste sea cero, sino, que dicha subsecuencia no está representada en la distribución.

El cálculo de la distancia coseno entre dos ejemplos se realiza mediante la ecuación 5.2.

$$distCos(x_k, x_p) = 1 - \frac{\sum_{j=1}^{n} x_{kj} x_{pj}}{\sqrt{\sum_{j=1}^{n} x_{kj}^2 \sum_{j=1}^{n} x_{pj}^2}} \tag{5.2}$$

donde x_k y x_p son los dos vectores que se desean comparar y *n* representa la cantidad total de elementos diferentes. El elemento *i-ésimo* del vector x_k está representado por x_k^i.

Si se utiliza la distancia coseno como la medida de la distancia, el potencial se representa por la fórmula mostrada en la ecuación 5.3.

$$P(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1} distCos(x_k, x_i)}{k-1}} \tag{5.3}$$

La ecuación 5.3 muestra cómo para calcular el potencial de un ejemplo en el EC, es necesario almacenar los valores de todos los ejemplos representados en dicho espacio. Esto supone utilizar una gran cantidad de memoria y realizar un número elevado de operaciones, lo que dificulta la utilización de este enfoque en problemas que impliquen actuar en tiempo real.

Para resolver este problema, Angelov y Zhou [13] proponen una ecuación recursiva capaz de calcular el potencial de un ejemplo utilizando la distancia coseno sin necesidad de almacenar todos los ejemplos previamente procesados. En el Apéndice C se detalla la obtención de la expresión recursiva propuesta en [13]. La importancia de este cálculo reside en que el resultado de dicha expresión es exactamente el mismo que se obtiene si se almacenaran todos los ejemplos y se

aplicara la ecuación 5.3. La diferencia entre las dos expresiones es que la expresión recursiva sólo necesita almacenar un conjunto muy reducido de datos. La expresión recursiva del potencial es la siguiente:

$$P_k(x_k) = \frac{1}{2 - \frac{1}{k-1}\frac{1}{\sqrt{\sum_{j=1}^{n}(x_k^j)^2}}\sum_{j=1}^{n} x_k^j b_k^j} \; ; \; k = 2, 3, ... \; ; \; P_1(x_1) = 1 \qquad (5.4)$$

donde:

$$b_k^j = b_{(k-1)}^j + \sqrt{\frac{(x_k^j)^2}{\sum_{l=1}^{n}(x_k^l)^2}} \; ; \; b_1^j = \sqrt{\frac{(x_1^j)^2}{\sum_{l=1}^{n}(x_1^l)^2}} \; ; \; i = [1, n+1] \qquad (5.5)$$

Estas dos ecuaciones muestran cómo el potencial de un determinado ejemplo puede calcularse utilizando un vector de datos almacenado (b_k). El resultado del potencial de un nuevo ejemplo será el mismo utilizando el vector b_k en la ecuación 5.4, que utilizando todos los ejemplos anteriores mediante la ecuación 5.3. De esta forma, se reduce de forma drástica la cantidad de información necesaria para el cálculo de este valor.

ABCD-ADAP calcula el potencial de un ejemplo considerando la ecuación recursiva 5.4. Sin embargo, si los atributos de un determinado ejemplo pueden ser negativos, el potencial debe ser modificado. En este caso, el potencial de un ejemplo podría calcularse considerando el cuadrado de la distancia tal y como muestra la siguiente definición de potencial:

$$P(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1} distCos^2(x_k, x_i)}{k-1}} \qquad (5.6)$$

En el apéndice C se calcula la expresión recursiva del potencial considerando la ecuación 5.6. En este caso, la expresión recursiva necesita almacenar una cantidad de datos mucho mayor, lo que supone un decremento en la velocidad de cálculo y un incremento en la memoria necesaria. De esta manera, este cálculo demuestra que si los atributos de un ejemplo no son negativos, no es factible utilizar el potencial calculado como se indica en la ecuación 5.6 cuando el número de elementos de un ejemplo es medio o alto. Esta demostración se considera una a aportación a los SAA.

Actualización de los prototipos almacenados

BIBADAP inicialmente está vacía. El primero de los comportamientos observados pasará a ser el primero de los prototipos almacenados. El potencial de este primer ejemplo tendrá el valor 1.

Conforme se creen nuevos modelos a partir de las subsecuencias, se deben actualizar los potenciales de todos los prototipos almacenados en BIBADAP utilizando la ecuación recursiva 5.4. Esta actualización es necesaria porque cada nuevo ejemplo hace que la densidad de los ejemplos representados en el EC se modifique.

Inserción del nuevo ejemplo como prototipo

Para actualizar la biblioteca ABCD-ADAP, se crean y eliminan prototipos en función de los ejemplos que se van recibiendo. Se crea un nuevo prototipo si el ejemplo recibido representa mejor que el prototipo actual a un determinado grupo de ejemplos de una clase determinada. Para ello, se calcula recursivamente el potencial del nuevo ejemplo (x_k) con respecto a cada uno de los grupos existentes representados por los prototipos en BIBADAP. Este potencial se compara en cada grupo con su prototipo actual; si el nuevo potencial es mayor significa que el nuevo ejemplo representa mejor al grupo de ejemplos y por lo tanto pasará a ser un prototipo. Es decir, si se cumple la ecuación 5.7 se insertará un nuevo prototipo.

$$\exists i,\ i = [1, NumPrototipos] :\ P(x_k) > P(Prototipo_i) \tag{5.7}$$

donde x_k es el nuevo ejemplo recibido; $Prototipo_i$ representa el prototipo almacenado en i-ésima posición y $NumPrototipos$ indica el número de prototipos existentes. De esta manera, sólo se añaden a la biblioteca aquellos ejemplos que son representativos de una parte de los ejemplos analizados.

Eliminación de prototipos existentes

Si se inserta un nuevo prototipo en la biblioteca, la dispersión de los ejemplos previamente analizados varía. Por esta razón, puede que sea necesario borrar uno o varios prototipos si estos son representados por el prototipo añadido. Para llevar a cabo esta tarea, se considera que las funciones de pertenencia que describen el grado de asociación de un ejemplo dado con un prototipo son de forma gausiana. La forma gausiana se caracteriza por su capacidad de generalización y cobertura de todo el EC. Este idea se representa en la ecuación 5.8 que muestra cuándo un prototipo debe ser eliminado.

$$\exists i,\ i = [1, NumPrototipos] :\ \mu_i(x_k) > e^{-1} \tag{5.8}$$

El grado de asociación entre un prototipo y un ejemplo dado considerando la distancia coseno se representa en la ecuación 5.9.

$$\mu_i(Ejemplo) = e^{-\frac{1}{2}[\frac{distCos(Ejemplo, Prototipo_i)}{\sigma_i}]} ;\ i = [1, NumPrototipos] \tag{5.9}$$

donde $distCos(Ejemplo, Prototipo_i)$ es la distancia coseno entre un *ejemplo* y el $prototipo_i$; σ_i representa la dispersión de la función de asociación; es decir, el radio de la zona de influencia del $Prototipo_i$ [12]. Almacenado un único ejemplo, el valor de σ_0 es 1. La ecuación 5.10 muestra cómo se obtiene el valor de la dispersión de la función de asociación para el ejemplo *k-ésimo* (x_k).

$$\sigma_i(k) = \sqrt{\frac{1}{k}\sum_{j=1}^{k} cosDist(Prototipo_i, x_k)} \; ; \;\; \sigma_i(0) = 1 \tag{5.10}$$

donde *k* es el número de ejemplos en un mismo grupo. Este valor se obtiene a partir de los ejemplos de un mismo grupo (aquellos representados por el mismo prototipo) dado que dicho valor sólo afecta a un grupo determinado.

Sin embargo, al igual que ocurría con el cálculo del potencial, en este caso, la suma que se realiza en la ecuación 5.10 hace necesario almacenar todos los ejemplos previamente analizados. Para solucionar este problema, el valor de la dispersión de la función de asociación (σ_i), se puede actualizar de forma recursiva como se indica en [14], con lo que se almacenan muy poco datos de cada ejemplo y el resultado final es el mismo. La ecuación recursiva se muestra a continuación (Ecuación 5.11).

$$\sigma_i(k) = \sqrt{[\sigma_i(k-1)]^2 + \frac{1}{k}[cosDist^2(Prot_i, z_k) - [\sigma_i(k-1)]^2]} \tag{5.11}$$

Clasificación del nuevo ejemplo

La clasificación consiste en comparar el ejemplo observado con todos los prototipos almacenados en BIBADAP. Dicho ejemplo será clasificado dentro del grupo representado por el prototipo al que más se asemeja. Para ello, se calcula la distancia del ejemplo a cada uno de los prototipos existentes. Estos cálculos son realizados teniendo en cuenta la distancia coseno de forma que la clasificación de un nuevo ejemplo se realizará tal y como se muestra en la ecuación 5.12.

$$\begin{aligned} &Clase(x_z) = Clase(Prototipo^*); \\ &Prototipo^* = MIN_{i=1}^{NumPrototipos}(cosDist(x_{Prototipo_i}, x_z)) \end{aligned} \tag{5.12}$$

De esta manera, el ejemplo observado será clasificado como miembro del grupo a cuyo prototipo esté más cercano. Este método de clasificación difiere del propuesto en [13] porque en este caso no todos los elementos de un ejemplo tienen un valor asociado.

5.1.3. Clasificación vs. Agrupamiento

M-ADAP ha sido descrito como un proceso de clasificación que utiliza tantos EC como clases existentes en el dominio. Sin embargo, M-ADAP también puede ser utilizado para agrupar aquellos modelos de comportamientos similares de forma no supervisada, es decir, para realizar tareas de *clustering*. Para ello, el primer paso de ABCD-ADAP, la clasificación de un nuevo ejemplo, se inhibe en el algoritmo y un nuevo ejemplo únicamente actualiza el EC. Por lo tanto, sólo es necesario un EC en el que un prototipo almacenado en BIBADAP representa un *cluster* generado por el algoritmo.

5.1.4. Interpretación del clasificador

Considerando un nuevo ejemplo y todos los prototipos almacenados en BIBADAP, es posible interpretar la clasificación de éste mediante la utilización de reglas borrosas. Esta interpretación puede estar descrita por el tipo de reglas borrosas que se describen en la Figura 5.3.

Interpretación del clasificador

Reglas del tipo:

$Regla^i = SI\ (x_1\ es\ similar\ a\ x_1^{i*})\ Y\ (x_2\ es\ similar\ a\ x_2^{i*})\ Y\ ...$
$...\ Y\ (x_n\ es\ similar\ a\ x_n^{i*})\ ENTONCES\ (Grupo^i)$

donde:

X^{i*} = [x_1^{i*}, x_2^{i*}, ... , x_n^{i*}] es el vector de características del $Prototipo_i$ almacenado en la biblioteca.

X = [x_1, x_2, ... , x_n] es el vector de características observadas (ejemplo).

$Regla^i$ es la regla borrosa *i* donde *i* = [1,*N*] y *N* es el número de grupos existentes.

$(x_j\ es\ similar\ a\ x_j^{i*})$ denota si el valor x_j del ejemplo (*X*) es similar al valor x_j del prototipo *i* (*j=[1,n]* donde *n* es el número de características del vector).

$Clase^i$ es la etiqueta que representa la clase del prototipo *i*.

Figura 5.3: Interpretación del clasificador obtenido utilizando reglas borrosas.

5.2. Evaluación: Interfaz de línea de comandos

En este dominio, el objetivo es analizar el comportamiento de un usuario a partir de los comandos que éste ha tecleado durante un determinado periodo de tiempo en una línea de comandos. Sin embargo, a diferencia de los experimentos realizados en el proceso de evaluación de MAUSE (apartado 4.4) en donde se desea modelar el comportamiento de usuarios individuales, en este caso los usuarios están agrupados en cuatro clases en función del uso que le dan a la interfaz de línea de comandos. Aplicando M-ADAP, cada clase estará representada por uno o varios prototipos en su correspondiente EC.

5.2.1. Conjunto de Datos

Para evaluar M-ADAP en este dominio se ha utilizado un conjunto de datos que tiene los comandos tecleados a través de la línea de comandos durante un periodo largo de tiempo por 168 usuarios [83]. Estos usuarios están agrupados en clases.

Este conjunto de datos se ha utilizado en investigaciones relacionadas con la utilización de la línea de comandos [81, 82]. Saul Greenberg recopiló estos datos que están disponibles en su página web [84].

Las 4 clases en las que se agrupan los usuarios son:

- *Programadores principiantes:* los usuarios agrupados en esta categoría tienen pocos o nulos conocimientos sobre programación, sistemas operativos e interfaces basados en comandos. Estos usuarios utilizan la mayor parte del tiempo en el ordenador para aprender a programar y manejan las funciones básicas del sistema operativo.

- *Programadores expertos:* en esta categoría se encuentran aquellos estudiantes de grado que tienen algunos conocimientos sobre lenguajes de programación y el entorno Unix. Además de programar, utilizan procesadores de texto, y funciones más avanzadas del sistema operativo.

- *Científicos informáticos:* este grupo de usuarios incluye a los estudiantes de postgrado y los investigadores del departamento de informática que tienen mucha experiencia con el entorno Unix y con los ordenadores en general. Las tareas que ejecutan son menos predecibles y más variadas. Entre ellas, la búsqueda bibliografica, el desarrollo de programas avanzados, comunicaciones, mantenimiento de bases de datos, uso de procesadores de texto, etcétera.

- *No programadores:* las principales tareas que realizan los usuarios de este grupo es el uso del procesador de texto y la preparación de documentos. Este grupo está formado por los usuarios que se encuentran en las oficinas de la universidad y su conocimiento de Unix es mínimo.

La Tabla 5.1 muestra el número de usuarios de cada clase, el número total de líneas de comandos tecleadas por todos los usuarios de cada grupo y el número medio de comandos tecleados por un usuario en una sesión.

Tabla 5.1: Descripción del conjunto de datos utilizado.

Nombre de la clase	Número de usuarios	Nº de líneas de comandos	Media de número de comandos por sesión
Programadores Principiantes	55	77423	14,9
Programadores Expertos	36	74906	19,4
Científicos informáticos	52	125691	16,2
No programadores	25	25608	13,4
Total	168	303628	16,2

5.2.2. Aplicación de M-ADAP

En este apartado se detalla cómo un agente implementa M-ADAP en el dominio de la interfaz de comandos de línea de comandos utilizando el conjunto de datos explicado previamente. Al aplicar M-ADAP, lo primero que se realiza es obtener la secuencia de eventos que representa el comportamiento de un usuario mediante el MCSE. En esta experimentación, cada evento representa un comando introducido a la línea de comandos.

Tal y como se explica en [83], los datos utilizados en esta experimentación fueron recogidos mediante el intérprete de comandos *csh* [101]. Considerando el formato utilizado por *csh*, cada línea no vacía del fichero de traza fue analizada y clasificada con un código (letra) que indica lo siguiente:

```
S: Comienzo de una sesión.
E: Final de una sesión.
C: Línea de comando insertado por el usuario.
D: Directorio de trabajo actual.
A: Alias invocado por la línea de comando (podría ser nulo).
H: Indica si la línea de comando fue recuperada del historial.
X: Indica si ha ocurrido un error (estará seguida de una letra
```

que indique el tipo de error).

A modo de ejemplo, a continuación se muestra un pequeño fragmento de texto tal y como ha sido procesado. El texto entre corchetes indica los comentarios y no son parte del fichero:

```
S - Fri Feb 20 23:39:46 1987  [Comienzo de sesión]
E - NIL  [Tiempo de final de sesión no disponible]
C - who  [Línea de comando insertada en el intérprete de comandos (csh)]
D - /user/lib098/100056  [Directorio actual]
A - who | more  [«who» es un alias para «who | more»]
H - NIL  [Línea no recuperada del historial]
X - NIL  [La línea no ha generado código de error]
```

Para poder utilizar estos datos, el MCSE tiene que convertirlos en una secuencia de comandos. Para ello, selecciona únicamente aquellas líneas marcadas como comandos tecleados por el usuario (etiquetadas con *C*). En el siguiente ejemplo, se muestran los datos originales en la parte izquierda, y la secuencia de comandos generada por el MCSE.

```
S Wed May 13 20:54:22 1987      *SOS* [Comienzo de sesión]
E Wed May 13 22:58:29 1987      mail
C mail                          ls
D /user/srdg/xxxxx              ruptime
A NIL                           ...
H NIL
X NIL
C ls
D /user/srdg/xxxxx
A /bin/ls -Fs
H NIL
X NIL
C ruptime
...
```

Cabe señalar que el inicio de cada sesión es indicado por el evento **SOS** (*Start Of Session*) y el final por el evento **EOS** (*End Of Session*).

A partir de la secuencia de eventos (comandos) generada por el MCSE para un usuario, el MCMC genera su modelo de comportamiento representado como un *trie*. A continuación, el MGMA convierte el modelo generado en una distribución

de secuencias de comandos para poder aplicar el algoritmo ABCD-ADAP. Debido a la gran cantidad de datos utilizados para crear el modelo de un usuario, la distribución equivalente posee un número elevado de subsecuencias. Al terminar de tratar los ejemplos disponibles, la biblioteca BIBADAP contendrá un número de prototipos que dependerá de lo homogéneos que sean los datos de los usuarios agrupados en una misma clase. Cada una de las cuatro clases estará representada por uno o más de estos prototipos representados con su correspondiente EC. Así, en BibAdap se encuentran almacenados todos los prototipos identificados con la clase a la que pertenecen.

Con la finalidad de evaluar M-ADAP, los comandos disponibles por cada usuario se dividen en dos partes; con una de estas partes, denominada *secuencia de entrenamiento*, se inicializa y actualiza la biblioteca BIBADAP y con la otra, *secuencia de evaluación*, se evalúa el clasificador generado.

5.2.3. Configuración Experimental

En esta serie de experimentaciones se utiliza *validación cruzada* con 10 particiones. Como el número de comandos que se utilizan para construir los modelos influye en los resultados de la clasificación, se han realizado experimentos con secuencias de distintas longitudes (100, 500 y 1.000 comandos). Para esta selección, se tienen en cuenta los últimos comandos del fichero de un usuario.

Dado que para obtener el modelo de comportamiento de un usuario es necesario generar el *trie* que representa utilizando el MCMC, es preciso definir el valor de la longitud de subsecuencia. Considerando los resultados obtenidos en la evaluación de MAUSE en este dominio (apartado 4.4), en estos experimentos se utilizan longitudes de entre 2 y 6 comandos.

Cabe señalar que en este dominio el número de subsecuencias diferentes que se obtienen es muy elevado. La Tabla 5.2 refleja cómo aumenta este número en función de la secuencia de entrenamiento y la longitud de las subsecuencias.

Tabla 5.2: Número total de subsecuencias diferentes obtenidas.

Número de comandos por usuario	Longitud de subsecuencia	Número de subsecuencias diferentes
100	3	11451
	5	34164
500	3	39428
	5	134133
1.000	3	63375
	5	227715

A diferencia de muchos de los algoritmos de clasificación utilizados actualmente, ABCD-ADAP no necesita un proceso previo de configuración, sino que se adapta a los distintos ejemplos que observa. De este modo, la cantidad de prototipos que se crean por cada clase depende de la homogeneidad de los datos que los forman. La Tabla 5.3 muestra, a modo de ejemplo, la cantidad de prototipos generados para cada una de las cuatro clases considerando 1.000 comandos por usuario y con una longitud de subsecuencia de 5. Dado que para esta experimentación se utilizará validación cruzada con 10 particiones, el número de prototipos varía en cada partición. Se puede observar que el grupo *programadores principiantes* tiene 55 usuarios y el número de prototipos creados varía entre 2 y 5. Por otra parte, el grupo *no programadores* es más homogéneo por lo que para su modelado sólo es necesario un único prototipo.

Tabla 5.3: M-ADAP: Número de prototipos creados por clase utilizando validación cruzada con 10 particiones

	Partición									
Clase (nº de usuarios)	1	2	3	4	5	6	7	8	9	10
Programadores Principiantes (55)	4	5	3	4	4	3	2	3	4	5
Programadores Expertos (36)	1	1	1	1	2	3	2	1	1	2
Científicos informáticos (52)	1	1	1	1	1	1	2	2	2	2
No programadores (25)	1	1	1	1	1	1	1	1	1	1

Dado que la mayoría de los algoritmos de clasificación requieren que los ejemplos posean un número fijo de atributos y un valor para cada atributo, un usuario se representa como un vector que contiene tantos atributos como subsecuencias de comandos existan considerando todos los usuarios. Es decir, para poder comparar ABCD-ADAP con otros algoritmos, es necesario representar los modelos de agentes en una estructura atributo-valor equivalente a la distribución de comandos utilizada por ABCD-ADAP. Para ello, primero se obtienen todas las posibles subsecuencias considerando todos los usuarios y después a cada usuario se le asigna un valor para cada una de las subsecuencias en función de su frecuencia. Esto implica que todo ejemplo está representado por un número de atributos elevado. Como todos los ejemplos necesitan un valor para cada uno de los atributos, el valor de los atributos correspondientes a subsecuencias que no han sido tecleadas por el usuario será cero.

Se han utilizado dos tipos de algoritmos de clasificación; incrementales y no incrementales. A continuación se detallan estos algoritmos:

- *Incrementales:* Este tipo de algoritmos no necesita disponer de todos los datos *a priori*, sino que tratan los ejemplos a medida que los reciben. Los

algoritmos incrementales de clasificación utilizados para compararlos con ABCD-ADAP son el Naive Bayes Incremental y el Vecino más cercano implementado de forma incremental (k-NN Incremental).

- *No Incrementales:* Dado que en este dominio se dispone *a priori* de todos los datos, se ha decidido comparar ABCD-ADAP con este tipo de algoritmo. Sin embargo, una de las características principales de ABCD-ADAP es su cualidad de ser incremental por lo que los resultados que se obtienen en esta comparación son a título orientativo. Se han utilizado tres algoritmos no incrementales muy conocidos y utilizados en una gran variedad de dominios: C5.0 [154], Naive Bayes [60] y el Vecino más cercano [53].

Cada uno de estos algoritmos de clasificación se describe de forma breve a continuación:

- *Naive Bayes:*

  Algoritmo de clasificación basado en el teorema de Bayes [60] y que a pesar de su simplicidad es comparable con otros algoritmos más sofisticados como las redes de neuronas artificiales o los árboles de decisión. Este algoritmo considera que todas las variable de un ejemplo son condicionalmente independientes dado el valor de la clase. Es decir, la presencia o ausencia de una variable en una clase no está relacionada con la presencia o ausencia de cualquier otra variable.

  Por otra parte, este algoritmo necesita ajustar una serie de parámetros relacionados con el tipo de función de densidad de probabilidad de los atributos. Si estos parámetros son escogidos basándose en el análisis de todos los datos de entrenamiento, este algoritmo necesita disponer de ellos *a priori* por lo que será no incremental. Si por el contrario se utiliza un valor por defecto, el algoritmo de clasificación puede ejecutarse de forma incremental.

- *k-NN (k-vecino más cercano):*

  El algoritmo de clasificación Vecino Más Cercano, conocido también como k-NN (K-Nearest Neighbor) [53], utiliza un tipo de aprendizaje basado en instancias. Este algoritmo compara una nueva instancia con todas las observadas previamente y le asigna la clase de la instancia más similar. Para ello, utiliza la distancia Euclídea normalizada con la que detecta las instancias de entrenamiento más cercanas a la que se quiere clasificar. Este algoritmo necesita establecer inicialmente el valor de k, que indica el número de instancias más cercanas que se van a tener en cuenta para clasificar un ejemplo dentro de una clase determinada. En este caso, se ha determinado de forma experimental el valor k=1 por obtener con este valor los mejores resultados;

a pesar de que esto hace que el clasificador obtenido sea más sensible al ruido.

Aunque este algoritmo puede parecer similar a ABCD-ADAP, existen diferencias entre ellos. La base de casos utilizada por k-NN es normalmente de gran tamaño porque necesita almacenar todos los datos observados; ABCD-ADAP almacena únicamente una serie de prototipos y cierta información reducida para posteriores cálculos.

- *Algoritmo C5.0:*

  Este algoritmo es la versión comercial de C4.5 [154] y está basado en la creación de árboles de decisión a partir de una estrategia de *divide y vencerás* similar a la utilizada por el algoritmo *ID3* [27]. En el algoritmo *C5.0*, cada nodo del árbol está asociado con un conjunto de instancias a las que se les asignan ciertos pesos para tener en cuenta valores de atributos no conocidos.

5.2.4. Resultados

La Tabla 5.4 muestra el porcentaje de usuarios correctamente clasificados utilizando diferentes números de comandos para entrenamiento y validación (100, 500 y 1.000) y diferentes longitudes para la segmentación inicial de la secuencia de comandos (de 2 a 6 comandos).

Analizando estos resultados se observa que la tasa de aciertos obtenida por el algoritmo de clasificación utilizado en M-ADAP incrementa al aumentar tanto el número de comandos utilizados para el entrenamiento como la longitud de subsecuencias utilizadas.

Estos resultados muestran cómo el porcentaje de usuarios correctamente clasificados utilizando ABCD-ADAP, es mejor que el obtenido por *k-NN* tanto en su forma incremental como no incremental.

Si se comparan los resultados obtenidos por ABCD-ADAP y el algoritmo Naive Bayes (tanto incremental como no incremental), se observa que este último obtiene mejores resultados incluso utilizando un número elevado de comandos de entrenamiento. Esta diferencia es mayor si se utilizan subsecuencias pequeñas, de 2 o 3 comandos, sin embargo se decrementa conforme se aumenta dicha longitud (5 o 6 comandos).

Se quiere remarcar de nuevo, que la comparación de ABCD-ADAP con otros algoritmos de clasificación se realiza a título orientativo. Ninguno de los algoritmos con los que se comparara ABCD-ADAP cumple todas las características de éste, entre las que se encuentran que es incremental y que necesita almacenar muy poca información para su funcionamiento.

Tabla 5.4: Porcentaje de aciertos de diferentes algoritmos en la clasificación de usuarios Unix utilizando diferentes números de comandos por usuario para el entrenamiento y validación (#1) y diferentes longitudes de subsecuencia (#2)

		Clasificador utilizado y su tasa de aciertos (en %)					
			Clasific. Incrementales		Clasific. No Incrementales		
#1	#2	ABCD-ADAP	Naive Bayes Incremental	k-NN Incremental (k=1)	C5.0	Naive Bayes	k-NN (k=1)
100	2	65,5	77,3	38,3	73,9	79,1	42,8
	3	64,9	76,1	36,5	69,6	79,7	39,8
	4	64,5	72	34,1	74,6	74,4	39,2
	5	67,9	73,2	32,3	68,6	75	33,3
	6	64,3	73,2	32,3	70,1	77,3	32,7
500	2	58,3	77,9	33,9	73,9	82,1	41,9
	3	59,5	73,8	36,9	74,6	77,3	39,8
	4	59,2	70,8	39,2	73,9	76,7	38,9
	5	66,7	71,4	35,1	73,6	76,1	37,3
	6	70,8	72,6	35,7	75,6	75,5	36,9
1.000	2	60,1	78,5	43,6	73,9	85,1	44,1
	3	65,5	77,9	44,0	74,6	81,5	47,3
	4	61,3	77,3	43,5	73,9	79,1	46,1
	5	70,2	76,7	42,5	73,6	78,5	44,0
	6	72,0	76,1	41,9	74,6	77,9	44,6

Adaptación de los algoritmos a una nueva clase

Una característica importante de los algoritmos incrementales utilizados, es su capacidad de adaptarse a aquellos ejemplos pertenecientes a una nueva clase, es decir, una clase de la cual no se ha recibido previamente ningún ejemplo.

En este apartado se examina cómo los clasificadores creados por los diferentes algoritmos de clasificación se adaptan a una nueva clase. Utilizando validación cruzada con 10 particiones del mismo conjunto de datos, se compara ABCD-ADAP con los algoritmos utilizados previamente. Este proceso sigue los siguientes pasos:

1. Se elige una de las cuatro clases del conjunto de datos con la finalidad de comprobar la capacidad de los diferentes algoritmos de crear clasificadores que se adapten a dicha clase. Esta clase se nombra como *nueva clase*.

2. Inicialmente se crea un conjunto de datos de entrenamiento que contiene un sólo ejemplo de la *nueva clase* y todos los ejemplos disponibles de las otras

tres clases. Se crean cada uno de los cuatro clasificadores que se desean comparar utilizando este conjunto de datos de entrenamiento.

3. Se crea un conjunto de datos de test que contiene ejemplos de las cuatro clases de las que se dispone. Con estos datos se calcula el porcentaje de los ejemplos pertenecientes a la *nueva clase* que han sido clasificados correctamente.

4. El número de ejemplos de la *nueva clase* en el conjunto de datos de entrenamiento se incrementan en uno y se repite el proceso desde el punto 2. En cada iteración se calcula la tasa de aciertos sobre la *nueva clase*. Este proceso se repite hasta que todos los ejemplos disponibles son incluidos en el conjunto de datos de entrenamiento.

La Figura 5.4 muestra los resultados de este proceso considerando como *nueva clase* la clase *Programadores Principiantes*.

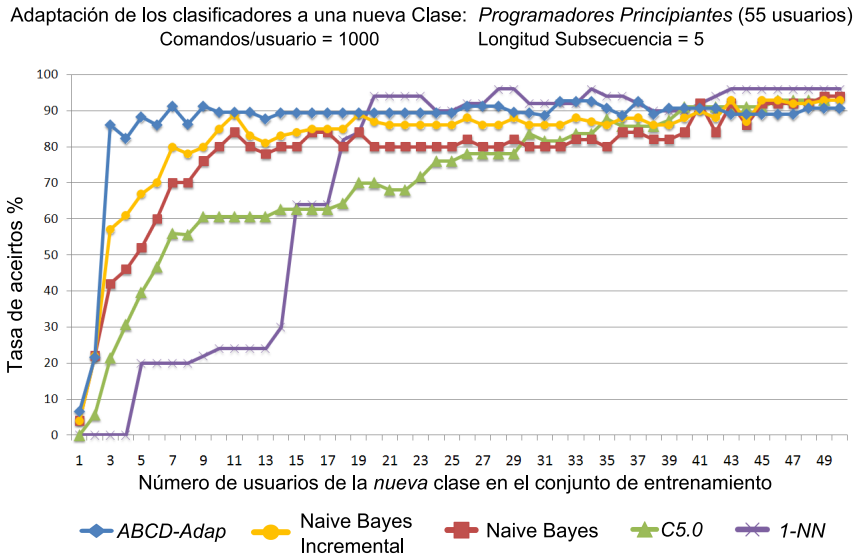


Figura 5.4: Tasa de aciertos de la *nueva clase* (*Programadores Principiantes*) incrementando el número de usuarios pertenecientes a dicha clase en el conjunto de entrenamiento.

En esta gráfica:

- El *eje x* representa el número de usuarios pertenecientes a la clase *Programadores principiantes* utilizados para entrenar cada uno de los clasificadores utilizados. Dado que esta clase contiene 55 usuarios y se ha utilizado validación cruzada con 10 particiones, la cantidad máxima de usuarios utilizados para entrenar los clasificadores es 49, el 90 % de los usuarios totales de la clase.

- El *eje y* representa el porcentaje de usuarios de la clase *Programadores principiantes* del conjunto de test, que han sido clasificados correctamente. Al utilizar validación cruzada, este valor es la media de los resultados obtenidos con cada una de las 10 particiones de test.

Como se puede ver en la gráfica, la estructura del clasificador creado por ABCD-ADAP se adapta rápidamente a la nueva clase. Únicamente con tres usuarios de esta clase, este algoritmo es capaz de crear uno o varios prototipos mediante el cual casi el 90 % de los usuarios (test) de dicha clase son clasificados correctamente. El incremento en la tasa de aciertos no es constante, se pueden observan picos debido a que un nuevo ejemplo puede generar, en ocasiones, cierto ruido.

Los algoritmos *Naive Bayes*, *Naive Bayes Incremental* y *C5.0* necesitan mayor cantidad de ejemplos de entrenamiento para obtener resultados buenos en la clasificación de ejemplos de esta *nueva clase*. Especialmente *C5.0* necesita recibir varios ejemplos de una misma clase para poder adaptar el árbol generado a la nueva clase. Por otro lado, cuando el clasificador *1-NN* obtiene buenos resultados se deben a que casi todos los ejemplos del conjunto de test los clasifica como de la nueva clase; lo que provoca una tasa de acierto muy baja para las otras clases, especialmente la clase *científicos informáticos*.

Las Figuras 5.5, 5.6 y 5.7 muestran los resultados para las clases *Programadores Expertos*, *Científicos informáticos* y *No Programadores*. Los resultados muestran que en todos los casos ABCD-ADAP requiere pocos ejemplos para adaptarse a una nueva clase.

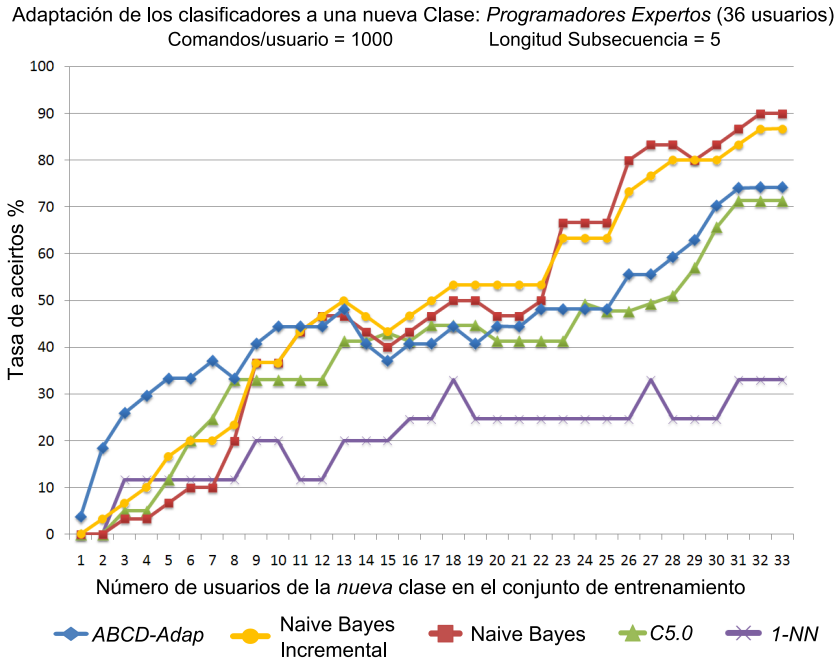


Figura 5.5: Tasa de aciertos de la *nueva clase* (*Programadores Expertos*) incrementando el número de usuarios pertenecientes a dicha clase en el conjunto de entrenamiento.

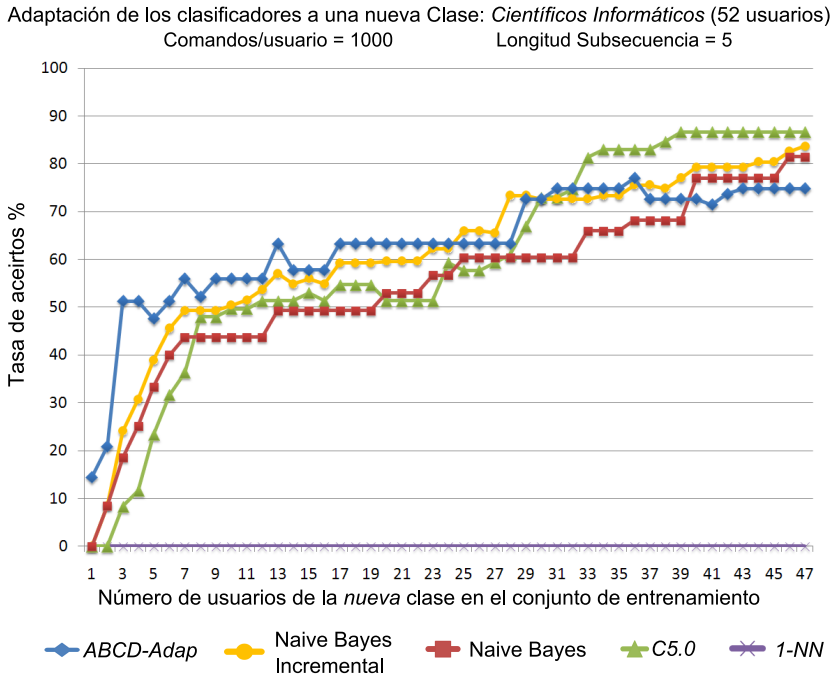


Figura 5.6: Tasa de aciertos de la *nueva clase* (*Científicos Informáticos*) incrementando el número de usuarios pertenecientes a dicha clase en el conjunto de entrenamiento.

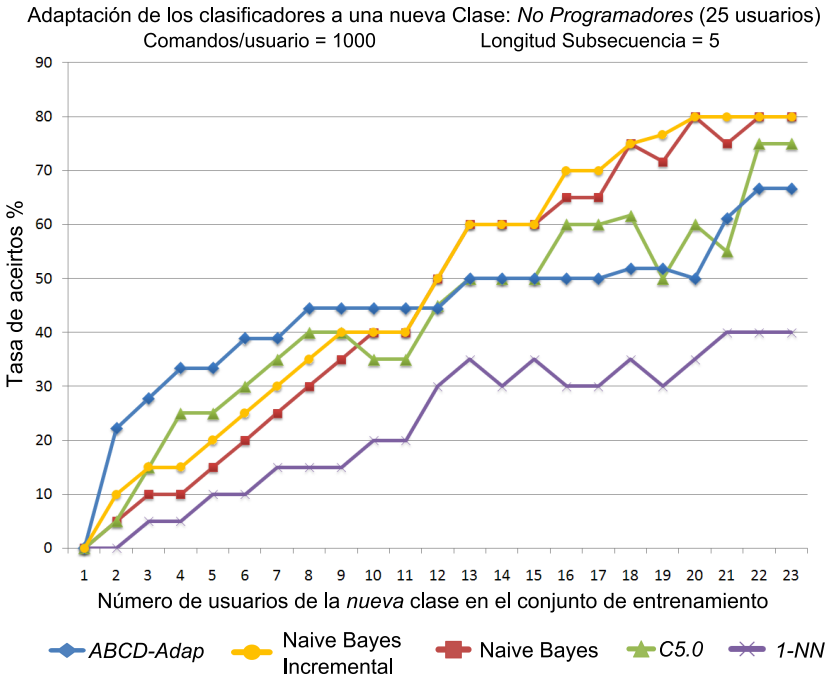


Figura 5.7: Tasa de aciertos de la *nueva clase* (*No Programadores*) incrementando el número de usuarios pertenecientes a dicha clase en el conjunto de entrenamiento.

5.3. Evaluación: Entornos inteligentes

Un entorno inteligente es un lugar (hogar, oficina, hospital, etc) equipado con una serie de sensores y actuadores capaces de realizar diferentes acciones para adaptarse a las preferencias de las personas en dicho lugar [192]. Esta tecnología puede aplicarse para tareas muy diversas. Por ejemplo, asistir a personas mayores o discapacitadas en el uso de dispositivos de la casa [137], en caídas u otros accidentes [160], etcétera. Actualmente el interés en este tipo de entornos está en continuo crecimiento, favorecido en parte por los recientes avances en tecnologías relacionadas con los diferentes tipos de sensores que ofrecen la oportunidad de monitorizar la actividad que realiza una persona. Sin embargo, todavía son necesarios otros avances tecnológicos para poder asistir a una persona de forma eficaz en este tipo de entornos. Uno de estos avances es el reconocimiento eficaz de las acciones realizadas por una persona.

En este tipo de entornos, un agente puede recoger mediante diferentes tipos de sensores, datos sobre la actividad que desarrolla una persona en la casa. La interpretación de los valores obtenidos de los diferentes sensores por parte del agente, permite modelar el comportamiento de una persona. Esta experimentación tiene como finalidad poder modelar y reconocer ciertas actividades que realiza una persona en función de los datos recibidos por los sensores con la que ésta interactúa.

5.3.1. Conjunto de Datos

Para poder desarrollar tecnologías que ayuden a la monitorización de actividades en entornos inteligentes, es necesario disponer de datos reales. Sin embargo, la recolección de datos en este tipo de entornos es un proceso complejo dado que requiere tanto de expertos como de recursos para diseñar e instalar los sensores, controladores, red de datos, etcétera [49]. Además, muchos de los datos que se recolectan en determinados entornos inteligentes se realiza de forma privada y no son accesibles a la comunidad científica.

Para esta experimentación se utiliza un conjunto de datos accesibles públicamente creado como parte del proyecto *CASAS* [152]. Estos datos han sido recolectados en un apartamento situado en el campus de la Universidad del Estado de Washington (Washington State University, WSU). Las actividades realizadas por los residentes del apartamento *inteligente* se perciben mediante sensores. A partir de la información percibida se actúa sobre el entorno con la finalidad de mejorar el confort, la seguridad y/o la productividad de las personas [144].

El apartamento *inteligente* consta de un salón, una cocina y un recibidor. Como se muestra en la Figura 5.8, dicho apartamento está equipado con 26 sensores de movimiento distribuidos por toda la casa con aproximadamente un metro de

separación (nombrados como M01, M02...M26). Además, están instalados los siguientes sensores:

- 8 sensores que detectan si los siguientes objetos son retirados de su posición habitual: bote de avena, bote de pasas, bote de azúcar, tazón de desayuno, cuchara, recipiente de medicinas, olla para cocinar y agenda telefónica. Estos sensores son nombrados como I01, I02, ... I08.
- Sensor de apertura del armario (D01).
- Sensor que detecta el uso de agua caliente y agua fría indicando además su temperatura (AD1-A y AD1-B).
- Sensor de uso de la cocina (AD1-C).
- Sensor que detecta cuándo se descuelga el teléfono (asterisk).

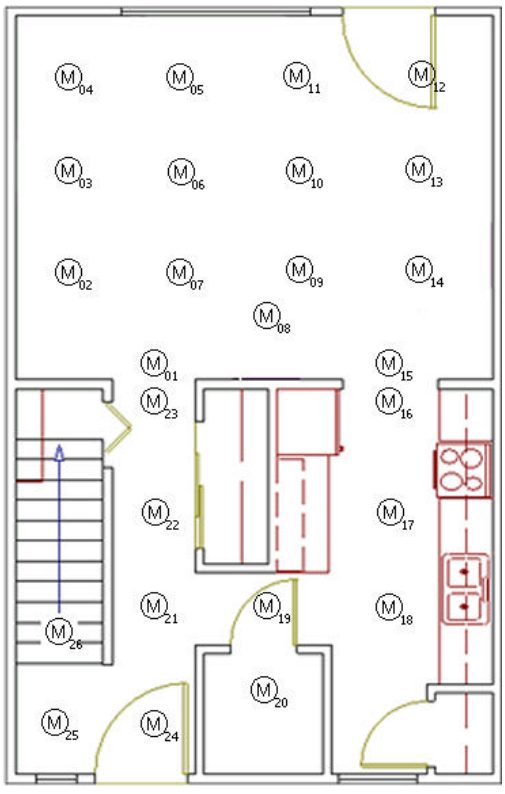


Figura 5.8: Plano del apartamento *inteligente* que incluye los sensores de movimiento utilizados en la toma de datos.

Para la recuperación de los datos, 24 estudiantes universitarios utilizaron el apartamento de forma individual realizando un conjunto de actividades cotidianas mientras los sensores recolectaban y almacenaban sus lecturas en una base de datos. La selección de actividades se realizó teniendo en cuenta aquellas acciones con más relevancia en la vida diaria y que son definidas en cuestionarios clínicos como *actividades funcionales diarias* [156]. Deficiencias en la realización en este tipo de actividades podrían identificar a aquellos individuos que están teniendo dificultades para vivir de forma independiente en un determinado entorno [167]. Las cinco acciones seleccionadas se detallan a continuación:

1. *Usar el teléfono:* esta actividad consiste en buscar un número telefónico en la agenda, realizar la llamada y escribir un mensaje en una libreta. La agenda telefónica, la libreta y el teléfono se encuentran en la mesa del salón.

2. *Lavarse las manos:* la realización de esta actividad supone lavarse las manos en el fregadero de la cocina utilizando jabón y toallas de papel.

3. *Cocinar:* esta actividad requiere calentar agua en la cocina y preparar avena con azúcar y pasas. Los materiales necesarios para esta tarea se encuentran en la encimera y en un determinado armario de la cocina.

4. *Comer y medicarse:* Para realizar esta actividad, se debe verter en un plato la avena cocinada, y llevarlo a la mesa del salón junto con un vaso de agua y el recipiente de medicinas. Una vez allí, además de comer se tomará la medicación.

5. *Limpiar:* esta actividad consiste en limpiar los platos en el fregadero de la cocina y dejar todos los utensilios utilizados en el armario correspondiente.

La toma de datos se llevó a cabo almacenando las lecturas de los sensores activados por cada uno de los 24 residentes que utilizaron el apartamento de forma individual realizando las cinco acciones indicadas. La lectura de un sensor incluye el identificador del propio sensor, el día y la hora en la que éste fue activado, y la lectura del mismo. Toda acción realizada por un usuario está representada por una lista ordenada de todos aquellos sensores que se han activado durante su realización. El conjunto de datos está formado por 120 ficheros que describen una de las cinco actividades realizada por uno de los 24 residentes y este conjunto tiene un total de 5.312 lecturas de sensores.

En la parte superior de la Figura 5.9 se muestra una persona realizando la actividad *"Lavarse las manos"* en el apartamento inteligente junto con los datos que proporcionan cinco sensores activados en esta actividad. La lectura de uno de los sensores (AD1-B) indica la temperatura del agua. La parte inferior de la Figura muestra la misma información recogida al realizar la actividad *"Cocinar"*.

Actividad: "Lavarse las manos"

Fecha / Hora	Id. del sensor	Lectura
2008-02-27 14:14:43.531	M17	OFF
2008-02-27 14:14:43.580	M17	ON
2008-02-27 14:14:44.740	M16	OFF
2008-02-27 14:14:47.498	AD1-B	0.03
2008-02-27 14:14:48.986	M17	OFF
...		

Actividad: "Cocinar"

Fecha / Hora	Id. del sensor	Lectura
2008-02-29 13:25:05.527	I01	ABSENT
2008-02-29 13:25:09.190	D01	CLOSE
2008-02-29 13:25:10.513	M17	ON
2008-02-29 13:25:11.979	I07	ABSENT
2008-02-29 13:25:13.869	M17	OFF
...		

Figura 5.9: Valores obtenidos por los sensores utilizados en la realización de dos actividades cotidianas: *"Lavarse las manos"* y *"Cocinar"*.

5.3.2. Aplicación de M-ADAP

El agente que implementa M-ADAP con la finalidad de modelar y detectar actividades de una persona en un entorno inteligente debe, primeramente, obtener la secuencia de eventos a partir de las diferentes lecturas de los sensores, utilizando para ello el MCSE. A continuación, se creará la distribución de eventos relevantes mediante el MCMC y por último el agente aplica el MGMA para clasificar la actividad y actualizar la biblioteca de modelos.

Cada uno de los 24 residente realiza una vez cada actividad, por lo que el conjunto de datos está formado por 120 ficheros que almacenan las lecturas de cada sensor activado y fecha y hora de cuando se realizó. En este caso, cada evento se corresponde con la activación de un determinado sensor. A modo de ejemplo, las Tablas 5.5 y 5.6 muestran las lecturas de los sensores obtenidas por un residente al realizar las actividades *"Lavarse las manos"* y *"Cocinar"* y la correspondiente secuencia de eventos obtenida por MCSE.

Tabla 5.5: Ejemplo de secuencia de eventos generada por MCSE en la actividad *"Lavarse las manos"*.

Actividad: *"Lavarse las manos"*	
Lectura de los sensores:	*Secuencia:*
2008-02-27 14:14:43.531 M17 OFF	*M17-OFF*
2008-02-27 14:14:43.580 M17 ON	*M17-ON*
2008-02-27 14:14:44.740 M16 OFF	*M16-OFF*
2008-02-27 14:14:47.498 AD1-B 0.03	*AD1-B*
2008-02-27 14:14:48.986 M17 OFF	*M17-OFF*
...	...

Tabla 5.6: Ejemplo de secuencia de eventos generada por MCSE en la actividad *"Cocinar"*.

Actividad: *"Cocinar"*	
Lectura de los sensores:	*Secuencia:*
2008-02-29 13:25:05.527 I01 ABSENT	*I01-ABSENT*
2008-02-29 13:25:09.190 D01 CLOSE	*D01-CLOSE*
2008-02-29 13:25:10.513 M17 ON	*M17-ON*
2008-02-29 13:25:11.979 I07 ABSENT	*I07-ABSENT*
2008-02-29 13:25:13.869 M17 OFF	*M17-OFF*
...	...

5.3.3. Configuración Experimental

En estos experimentos, los datos iniciales son divididos en *secuencia de entrenamiento* y *secuencias de evaluación*. Debido a la reducida cantidad de datos disponibles, se utiliza validación cruzada con 3 particiones.

En el dominio de la interfaz de línea de comandos utilizado en el apartado anterior, el conjunto de datos utilizado tenía secuencias de gran cantidad de eventos, por lo que para la evaluación de M-ADAP este conjunto se reducía a 100, 500 y 1.000 comandos. Sin embargo, en esto dominio, cada una de las 120 secuencias que representan una actividad en el conjunto de datos, contiene aproximadamente entre 50 y 100 valores de sensores, por lo que para estos experimentos se utilizan todos los datos disponibles.

Al igual que en el dominio anterior, los resultados dependen de la longitud de subsecuencia que se utilice para construir el modelo de la persona. En este caso, se han utilizado subsecuencias que varían entre 2 y 6 (lecturas de sensores). La Tabla 5.7 muestra cómo varía el número de subsecuencias diferentes generadas en función de la longitud de subsecuencia utilizada.

Tabla 5.7: Número total de subsecuencias diferentes generadas en función de la longitud de subsecuencia considerada

Longitud de subsecuencia	Número de subsecuencias diferentes
2	501
3	1732
4	3755
5	6389
6	9469

5.3.4. Resultados

Además de aplicar M-ADAP con el conjunto datos explicado, en esta experimentación también se compara ABCD-ADAP con los algoritmos de clasificación utilizados en el dominio anterior.

La Tabla 5.8 muestra la tasa de aciertos de los diferentes algoritmos de clasificación. En ella se observa que los clasificadores construidos utilizando inicialmente todos los datos (no incrementales) obtienen mejores resultados. Además, la mayoría de los clasificadores obtienen los mejores resultados con longitudes de secuencia reducidas. En este dominio, esto puede deberse a que la elevada velocidad a la que se activan los sensores hace que una misma actividad pueda producir la activación de los mismos sensores pero con modificaciones en su orden de activación. En este caso la búsqueda de patrones de comportamiento que definan una actividad es más compleja cuanto mayor es la secuencia de lecturas considerada. Incluso la tasa de aciertos del algoritmo *1-NN* es del 95 % cuando se utiliza una longitud de subsecuencia de 2.

Los resultados del algoritmo ABCD-ADAP son buenos si se tiene en cuenta que, a diferencia de otros algoritmos, ABCD-ADAP es incremental, necesita almacenar información muy reducida y se adapta muy rápidamente a los cambios de una persona en la forma de realizar una actividad.

5.3.5. Comparación de M-ADAP con otras representaciones

El reconocimiento de actividades se ha abordado por otros investigadores utilizando técnicas muy diversas [129, 149, 196]. En la experimentación anterior, se comparan diferentes algoritmos de clasificación pero siempre considerando la distribución de subsecuencias obtenida por el MCSE y el MCMC. Sin embargo, los datos obtenidos mediante los sensores pueden representarse de diversas formas lo que permite aplicar diferentes técnicas.

En este mismo dominio, Cook y Schmitter [48] representan las actividades

Tabla 5.8: Porcentaje de aciertos de diferentes algoritmos de clasificación en la clasificación de actividades utilizando diferentes longitudes de subsecuencia (#) para la creación del modelo

	Clasificador utilizado y su tasa de aciertos (en %)					
		Clasific. Incrementales		Clasific. No Incrementales		
#	ABCD-ADAP	Naive Bayes Incremental	k-NN Incremental (k=1)	C5.0	Naive Bayes	k-NN (k=1)
2	92,5	93,3	91,6	94,1	97,5	95
3	**94,2**	**88,3**	**81,5**	**95**	**97,5**	**86,6**
4	87,5	87,5	53,3	93,3	95,8	59,1
5	79,2	84,1	40,8	93,3	93,3	43,3
6	78,3	82,5	33,3	92,5	92,5	34,1

utilizando distintas características. Por ejemplo, el número de veces durante la actividad que el residente ha estado en una localización determinada, número de veces que se ha abierto el armario, abierto el grifo, encendido la cocina y utilizado el teléfono. Identificadas las acciones con estos valores, se aplican dos técnicas diferentes que se detallan a continuación.

- *Naive Bayes.* Se utiliza la frecuencia relativa de los valores de las características que identifican una actividad. Para ello, una secuencia de sensores se representa por el conjunto de valores indicados anteriormente y dicha secuencia (*D*) pertenecerá a la clase *a* (del conjunto de clases *A*) si cumple: $argmax_{a \in A} P(a|D) = \frac{P(D|a)P(a)}{P(D)}$. Considerando la configuración experimental propuesta en este apartado, la tasa de aciertos utilizando este clasificador y validación cruzada con 3 particiones es del 91 %.

- *Modelos de Markov.* En este caso, para cada actividad se crea un modelo de Markov, en el que el estado actual depende únicamente del estado previo. Las transiciones entre estados están definidas por probabilidades. La Figura 5.10 muestra un ejemplo de este tipo de modelos. Utilizando estos modelos, la tasa de acierto se eleva al 98 %.

Estos resultados muestran que en este dominio, la representación de los datos influye en la precisión de los clasificadores. En este caso, se observa que utilizando Naive Bayes y la representación definida por Cook y Schmitter, se obtiene un resultado del 91 %. Si utilizamos este mismo algoritmo con la representación utilizada en M-ADAP (con una longitud de subsecuencia de 3), el porcentaje de aciertos se eleva a un 97,5 %.

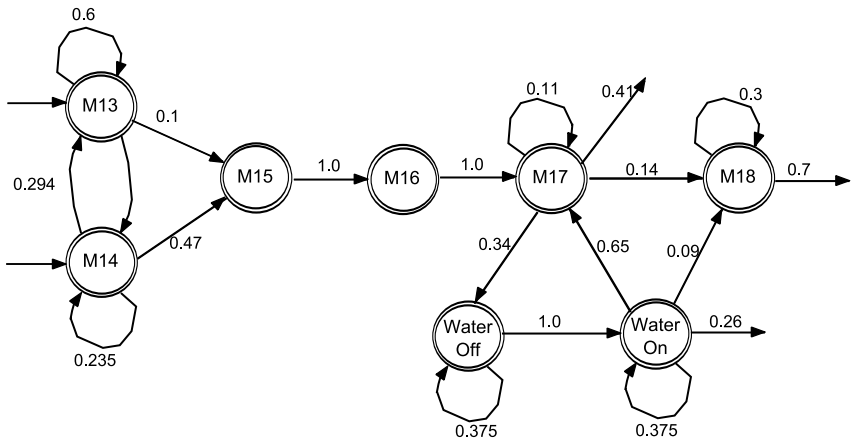


Figura 5.10: Modelo de Markov que representa la actividad: *"Lavarse las manos"*.

5.4. Conclusiones

En este capítulo se ha presentado un nuevo enfoque para el modelado de agentes basado en Sistemas Auto-Adaptativos, que en la actualidad se utilizan con finalidades muy distintas a la planteada en esta tesis doctoral. Aplicando *SAA* en el modelado de agentes, se resuelve en gran medida uno de los principales retos en este campo, la actualización continua de los modelos creados. El algoritmo de clasificación propuesto en este capítulo, mantiene la biblioteca de modelos constantemente actualizada en función de las observaciones.

Al igual que ocurría en los enfoques anteriores, el único parámetro que se define en M-ADAP es la longitud en la que se divide la secuencia de eventos. El valor más adecuado para este valor dependerá del dominio.

Esta experimentación también refleja las ventajas que tiene el algoritmo de clasificación propuesto frente a otros algoritmos de clasificación. Estas ventajas se resumen a continuación:

- ABCD-ADAP es capaz de gestionar un número elevado de modelos de comportamiento porque trata ejemplo por ejemplo, sin necesidad de almacenar y considerar todos los ejemplos para crear el clasificador. Esta ventaja hace que se puedan utilizar ejemplos cuya cantidad de atributos es muy elevada.

- ABCD-ADAP es incremental, por lo que no necesita disponer *a priori* de todos los datos, sino que trata los ejemplos a media que los recibe.

- ABCD-ADAP es capaz de adaptarse de forma rápida a cambios bruscos que se realicen en los datos. Con pocos ejemplos de una nueva clase es posible crear uno o varios prototipos que la representen.

- A diferencia de otros algoritmos, ABCD-ADAP no necesita definir ningún umbral, los valores del propio clasificador cambian en función de las distribuciones obtenidas.

Por último, en este capítulo también se ha expuesto como aportación a los SAA la expresión recursiva del cálculo del potencial considerando una modificación de éste. Dicha aportación es detallada en el Apéndice C y demuestra que la utilización del potencial considerando el cuadrado de la distancia coseno sólo es factible si los ejemplos que se traten contienen un número muy reducido de características diferentes.

Capítulo 6

Conclusiones Generales

6.1. Sumario

En esta tesis doctoral se han presentado y evaluado tres enfoques para el modelado y detección del comportamiento de agentes inteligentes. Estos enfoques son utilizados por un agente que obtiene observaciones del entorno en el que se encuentran los agentes que se desean modelar. Las diferencias en la finalidad de cada enfoque y los diferentes tipos de dominios en los que se puede aplicar cada uno de ellos, hacen que esta tesis aborde de forma amplia la tarea de modelar automáticamente el comportamiento de agentes.

Los tres enfoques propuestos están basados en el análisis de las acciones realizadas por un agente, considerando relevante el orden en el que éstas se han realizado. Para ello, todo comportamiento se representa por una secuencia de eventos (acciones) que se analiza de forma estadística con la finalidad de obtener aquellas subsecuencias de eventos más relevantes. La relevancia de estas subsecuencias se calcula utilizando dos métodos: los basados en frecuencia y los de dependencia estadística.

El primero de estos enfoques, M-COMP, se centra en la resolución de la problemática planteada en la competición *Coach 2006*. Su finalidad consiste en detectar y modelar el comportamiento de un conjunto de agentes a partir de su observación. En este caso, dicho modelo de comportamiento se obtiene a partir de la observación de dos comportamientos diferentes y su posterior comparación. La detección en tiempo real de un comportamiento previamente analizado también es parte de este enfoque.

MAUSE, el segundo de los enfoques propuestos, representa un modelo de comportamiento de agentes como una distribución de subsecuencias de eventos relevantes. Este enfoque también propone un clasificador basado en métodos estadísticos capaz de clasificar el comportamiento observado de uno o varios agentes en

uno de los modelos o perfiles previamente creados. Su generalidad hace que pueda aplicarse en dominios muy diferentes.

El tercer enfoque propuesto, denominado M-ADAP, propone un modelado de agentes mediante el flujo continuo de datos. A diferencia de los anteriores, este enfoque utiliza las observaciones del comportamiento de los agentes tanto para detectar su comportamiento como para actualizar los modelos ya creados. Este nuevo enfoque está motivado por la necesidad de adaptar los modelos creados a las variaciones que sufren los agentes en su comportamiento en determinados dominios. Para ello, se utiliza un algoritmo basado en *Sistemas Auto-Adaptativos* capaz de crear modelos que pueden ser actualizados y que representan comportamientos generales de grupos de agentes.

6.2. Conclusiones

El modelado y la detección del comportamiento de agentes inteligentes son tareas en continua y rápida expansión. Como se ha mostrado en el capítulo 2, los métodos de modelado propuestos en los últimos años y los dominios en los que éstos se han aplicado, son numerosos y diversos. Sin embargo, a pesar de las ventajas que proporciona el conocer cómo se comportan uno o varios agentes en un determinado entorno, definir un modelado general de comportamiento es una tarea compleja. En la mayoría de las investigaciones actuales, se proponen modelados no generales de un determinado tipo de agentes en un dominio concreto, es decir, modelados *ad hoc*.

El objetivo principal de esta tesis doctoral consistía en definir un marco que permitiera crear modelos de comportamiento de agentes inteligentes a partir de la observación de su comportamiento, y la posterior clasificación de un comportamiento observado como uno de los modelos previamente creados. Debido a las diferencias en las características particulares de todo dominio, para conseguir este objetivo se han propuesto, desarrollado y evaluado tres enfoques diferentes. Un agente utilizará uno u otro enfoque en función de las características del dominio en el que se encuentran los agentes que se quieren modelar.

Los tres enfoques presentados en esta tesis doctoral tienen en común que el comportamiento observado de los agentes que se desea modelar se transforma en una secuencia de eventos. La creación de esta secuencia a partir de las observaciones de dichos agentes es dependiente del dominio. Además, los enfoques planteados necesitan determinar la longitud de las subsecuencias en las que se segmenta la secuencia de eventos generada. Este parámetro representa la cantidad de eventos dependientes por lo que su valor óptimo, es decir, definir cuántos eventos son dependientes entre sí, varía en función del dominio y debe calcularse de forma experimental.

En un SMA, el comportamiento de un agente puede ser interferido por otros agentes. Esto hace compleja la tarea de modelar y detectar el comportamiento de uno o varios agentes en este tipo de dominios. A pesar de ello, los enfoques planteados en esta tesis doctoral son evaluados en dominios complejos como el fútbol simulado.

Con la finalidad de conseguir el objetivo planteado en dominios en los que se dispone de dos comportamientos diferentes de un mismo agente o conjunto de agentes, se ha diseñado y desarrollado M-COMP. M-COMP permite detectar diferencias en el comportamiento de agentes. Este enfoque es el único presentado en esta tesis doctoral que necesita definir una serie de umbrales que deben ser ajustados de forma experimental.

El modelo de comportamiento de uno o varios agentes obtenido mediante la aplicación de M-COMP está directamente relacionado con el tipo de eventos que forman la secuencia que representa su comportamiento. En el dominio del fútbol simulado, el reconocimiento de un patrón de comportamiento depende de dos factores, los eventos futbolísticos involucrados en un patrón y el número de veces que se ejecuta dicho patrón.

La evaluación de este enfoque en la competición *Coach 2006*, muestra la complejidad del objetivo propuesto. A pesar de ello, el análisis exhaustivo realizado de dicha evaluación muestra su viabilidad y, en general, buenos resultados.

El segundo de los enfoques planteados, denominado MAUSE, puede ser utilizado por un agente en dominios en los que el modelo de uno o varios agentes debe ser creado a partir de la observación de un único comportamiento. La generalidad del enfoque en este tipo de dominios se muestra aplicándose en dos entornos muy diferentes tanto en sus características como en su finalidad.

En el dominio del fútbol simulado, MAUSE se ha aplicado para la detección de patrones de comportamiento tal y como propone la competición *Coach 2006*. A diferencia de M-COMP, MAUSE no necesita definir ningún umbral y sus resultados son mejores que los obtenidos con el primero de los enfoques y similares o incluso mejores que los obtenidos por el campeón de dicha competición. En este dominio, al igual que ocurría con el enfoque anterior, cuando los eventos detectados no capturan la información relevante en los patrones de comportamiento de los agentes, éstos no son reconocidos.

En el dominio presentado como interfaz de línea de comandos, los resultados obtenidos al aplicar MAUSE en el modelado y detección de usuarios se han comparado con los obtenidos mediante la aplicación de HMM. Esta comparación refleja el buen funcionamiento del enfoque propuesto en este dominio incluso cuando la cantidad de comandos de los que se dispone para crear el modelo de un usuario es reducida. En este dominio, determinar una longitud de subsecuencia alta supone considerar que la repetición de muchos eventos en un mismo orden es

muy relevante. Sin embargo, esto no es así porque los usuarios no suelen repetir con exactitud una secuencia de muchos comandos. Considerar un valor alto para dicha longitud en este dominio, implica incorporar ruido en los datos.

El tercer enfoque propuesto, M-ADAP, afronta un importante reto en el modelado de agentes, la actualización de los modelos de comportamiento en función de los cambios que sufre el comportamiento de dichos agentes. Este enfoque se propone con la finalidad de realizar la tarea de modelado de agentes en dominios en los cuales los agentes modifican su comportamiento a lo largo del tiempo.

El enfoque propuesto, basado en Sistemas Auto-Adaptativos, utiliza una biblioteca de modelos que no necesita configuración previa y que está en constante actualización de forma automática. Una de las principales ventajas de este enfoque es que para aplicar el método de clasificación propuesto, no es necesario tener almacenados todos los ejemplos. Aquellos ejemplos que no forman parte de la biblioteca, se analizan, se actualizan determinadas variables con su información relevante, y son eliminados.

Este enfoque se ha evaluado en el dominio de la interfaz de línea de comandos y en la detección de actividades en entornos inteligentes. La experimentación llevada a cabo en estos dos dominios diferentes muestra que existen algoritmos de clasificación que obtienen mejores resultados que los obtenidos por el algoritmo de clasificación utilizado en M-ADAP. Sin embargo, M-ADAP es capaz de gestionar un número elevado de modelos de ejemplos porque no necesita almacenar todos los ejemplos. Además, el algoritmo de clasificación que utiliza este enfoque es incremental, por lo que no necesita disponer *a priori* de todos los datos, sino que trata los ejemplos en el momento que los recibe. También se ha demostrado la capacidad de M-ADAP para detectar rápidamente los cambios en los comportamientos observados y adaptarse así rápidamente al reconocimiento de nuevos comportamientos.

Uno de los objetivos que se deseaba alcanzar en esta tesis doctoral era crear un enfoque lo más general posible para el modelado automático de agentes. Los tres enfoques presentados varían en función de las características del dominio en el que éstos se implementen y la finalidad que se desea alcanzar con ellos. Sin embargo, estos enfoques son totalmente generalizables para el modelado y detección de comportamientos que puedan ser representados como una secuencia de eventos. Por lo tanto, estos enfoques pueden ser utilizados en dominios diferentes a los utilizados para su evaluación en esta tesis doctoral, siempre y cuando compartan características.

Por último, también se ha expuesto como aportación a los SAA la expresión recursiva del potencial considerándose una modificación de éste para que pueda ser utilizado cuando los atributos de los ejemplos pueden tener valores negativos.

6.3. Trabajos futuros

Los enfoques propuestos en esta tesis doctoral representan un comportamiento como una secuencia de eventos. Existen dominios en los que se puede obtener más información sobre un evento, por ejemplo, la duración de éste. Esta información adicional puede ser considerada en futuras líneas de trabajo. En este caso, el modelo de un usuario también almacenaría este dato y todo evento estaría definido no sólo por su orden en la secuencia, sino también por el tiempo durante el cual éste se ha realizado.

Quizás la futura línea de investigación más importante en el trabajo de esta tesis doctoral consiste en utilizar de forma eficaz la detección de comportamientos obtenida por los distintos enfoques propuestos. Por ejemplo, en el dominio de la *RoboCup*, detectar el comportamiento del oponente implica poder predecir sus acciones y generar las estrategias apropiadas para jugar contra éste. Esta tarea puede realizarse en dicho dominio utilizando el agente especial *coach*. Como ya se ha comentado, este agente puede observar todo lo que ocurre en el campo y comunicarse con los jugadores de su equipo. Así, el *coach* podría detectar el comportamiento del oponente utilizando los enfoques M-COMP o MAUSE y enviar a sus jugadores mensajes con la estrategia que éstos deben seguir. Como trabajo preliminar, en [95] se presenta una técnica para generar contra-estrategias automáticamente a partir de la detección del comportamiento del oponente. Para ello, en función del tipo de comportamiento reconocido, se crea la contra-estrategia (formada por una serie de acciones) que se debe llevar a cabo. A pesar de que la evaluación de esta propuesta no es sencilla, los primeros resultados son prometedores.

Por otra parte, en determinados dominios, las subsecuencias de eventos más *antiguas* son menos relevantes en el modelo creado. Por este motivo, se propone como trabajo futuro la actualización de los modelos de comportamiento mediante una técnica utilizada en [13] para actualizar la base de reglas de un sistema. Aplicando esta técnica en cada uno de los tres enfoques propuestos, cada subsecuencia de eventos se indexa con un número que indica el *momento* en el que se han realizado el primero de los eventos que la forman. Este valor se representa como un número entero desde uno hasta el total de subsecuencias obtenidas. Mediante este valor, se puede calcular el término definido como *edad* de una subsecuencia, que indica cómo de *antigua* es dicha subsecuencia en el modelo. Cómo se calcula este valor se muestra en la ecuación 6.1.

$$edad_S(t) = t - \frac{\sum_{i=1}^{N_s(t)} I_s(i)}{N_s(t)} \tag{6.1}$$

donde *t* es el instante de tiempo actual, *s* representa una subsecuencia determinada, $edad_s(t)$ denota la *edad* de la subsecuencia *s* en el momento *t*, $N_s(t)$ es el número de veces que la subsecuencia *s* fue observada hasta el momento *t* e $I_s(i)$ denota el momento de la subsecuencia *s* en el que ésta fue leída por *i-ésima* vez.

Usando este valor, se pueden actualizar los modelos almacenados en la biblioteca eliminándose aquellas subsecuencias más *antiguas*. Por ejemplo, una subsecuencia podría ser eliminada de un modelo si su *edad* es mayor que la media de *edad* de todas las subsecuencias. Esta media de *edad* puede calcularse tal y como se indica en la ecuación 6.2.

$$Edad(t) = \frac{1}{R}\sum_{j=1}^{R} edad_i(t) \tag{6.2}$$

Por último, se quiere resaltar de nuevo la cantidad y variedad de dominios en los que se pueden aplicar los enfoques propuestos. Además de diferentes dominios utilizados en esta tesis doctoral, se propone como línea de trabajo futura la aplicación de estos enfoques en otros dominios. Por ejemplo, el modelado y la clasificación de usuarios a partir de cómo utilizan un navegador de Internet [83], la detección de intrusos en un sistema informático [163] o la clasificación de secuencias biológicas [140].

6.4. Publicaciones

Esta tesis doctoral ha dado origen a las publicaciones que se muestran a continuación.

Título:	*CAOS Coach 2006 Simulation Team: An Opponent Modelling Approach*
Autores:	José Antonio Iglesias, Agapito Ledezma y Araceli Sanchis
Revista:	Computers and Informatics Journal
Publicación:	ISSN 1335-9150. Volumen 28, 2009, número 1, páginas: 57-80
Fecha:	Enero, 2009

Libro:	Soft Computing Methods for Practical Environmental Solutions: Techniques and Studies
Capítulo:	*User Modeling in Soft Computing Framework*
Autores:	José Antonio Iglesias, Agapito Ledezma y Araceli Sanchis
Editores:	Marcos Gestal Pose y Daniel Rivero Cebrián
Editorial:	IGI Global
Lugar:	Hershey, EEUU
Fecha:	Enero, 2010

Título:	*Winning Advantage: Using Opponent Models in Robot Soccer*
Autores:	José Antonio Iglesias, Juan Antonio Fernández, Ignacio Ramón Villena, Agapito Ledezma y Araceli Sanchis
Congreso:	International Conference on Intelligent Data Engineering and Automated Learning 2009, IDEAL 2009
Publicación:	Actas de IDEAL 2009
Lugar:	Burgos, España
Fecha:	Septiembre, 2009

Título:	*Creating User Profiles From a Command-Line Interface: A Statistical Approach*
Autores:	José Antonio Iglesias, Agapito Ledezma y Araceli Sanchis
Congreso:	International Conference on User Modeling, Adaptation, and Personalization, UMAP 2009
Publicación:	Actas de UMAP 2009
Lugar:	Trento, Italia
Fecha:	Junio, 2009

	Best Paper Award
Título:	*Modelling Evolving User Behaviours*
Autores:	José Antonio Iglesias, Plamen Angelov, Agapito Ledezma y Araceli Sanchis
Congreso:	IEEE Symposium Series on Computational Intelligence. IEEE Workshop on Evolving and Self-Developing Intelligent Systems, ESDIS 2009
Publicación:	Actas de ESDIS 2009
Lugar:	Nashville, TN, USA
Fecha:	Abril, 2009

Título:	*Using Well-Known Techniques for Classifying User Behavior Profiles*
Autores:	José Antonio Iglesias, Agapito Ledezma y Araceli Sanchis
Congreso:	International Workshop on Distributed Agent-based Retrieval Tools, DART 2008
Revista:	International Journal Communications of SIWN
Lugar:	Cagliari, Italia
Fecha:	Septiembre, 2008

Título:	*Classifying Efficiently the Behavior of a Soccer Team*
Autores:	José Antonio Iglesias, Agapito Ledezma, Araceli Sanchis y Gal Kaminka
Congreso:	International Conference on Intelligent Autonomous Systems, IAS-10 2008
Publicación:	Actas de IAS 2008
Lugar:	Baden Baden, Alemania
Fecha:	Julio, 2008

Título:	*An Efficient Behavior Classifier based on Distributions of Relevant Events*
Autores:	José Antonio Iglesias, Agapito Ledezma, Araceli Sanchis y Gal Kaminka
Congreso:	European Conference on Artificial Intelligence, ECAI 2008
Publicación:	Actas de ECAI 2008
Lugar:	Patras, Grecia
Fecha:	Julio, 2008

Título:	*Verifying Robocup Teams*
Autores:	Clara Benac Earle, Lars-Ake Fredlund, José Antonio Iglesias y Agapito Ledezma
Workshop:	International Workshop on Model Checking and Artificial Intelligence, MoChArt 2008
Publicación:	Actas del Workshop MoChArt 2008
Lugar:	Patras, Grecia
Fecha:	Julio, 2008

Título:	*Sequence Classification using Statistical Pattern Recognition*
Autores:	José Antonio Iglesias, Agapito Ledezma y Araceli Sanchis
Congreso:	International Conference on Intelligent Data Analysis 2007. IDA 2007
Publicación:	Actas de IDA 2007
Lugar:	Ljubljana, Eslovenia
Fecha:	Septiembre, 2007

Título:	*Caos Online Coach 2006 Team Description*
Autores:	José Antonio Iglesias, Agapito Ledezma y Araceli Sanchis
Congreso:	RoboCup 2006
Publicación:	CD RoboCup 2006
Lugar:	Bremen, Alemania
Fecha:	Junio, 2006

Título:	*A comparing method of two team behaviours in the Simulation Coach Competition*
Autores:	José Antonio Iglesias, Agapito Ledezma y Araceli Sanchis
Congreso:	International Conference on Modeling Decisions for AI, MDAI 2006
Publicación:	Actas MDAI 2006
Lugar:	Tarragona, España
Fecha:	Abril, 2006

Título:	Comparing Behavior in Agent Modelling Task
Autores:	José Antonio Iglesias, Agapito Ledezma, Araceli Sanchis
Congreso:	International Conference IADIS, Applied Computing 2006
Publicación:	Actas de IADS 2006
Lugar:	San Sebastián, España
Fecha:	Febrero, 2006

Bibliografía

[1] A. Aamodt and E. Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] K. T. Abou-Moustafa, M. Cheriet, and C. Y. Suen. On the structure of hidden markov models. *Pattern Recognition Letters*, 25(8):923–931, 2004.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering*, pages 3–14, 1995.

[4] M. Ahmadi, A. Keighobadi-Lamjiri, M. M. Nevisi, J. Habibi, and K. Badie. Using a two-layered case-based reasoning for prediction in soccer coach. In *Proceedings of the International Conference of Machine Learning; Models, Technologies and Applications (MLMTA-03)*, pages 181–185, 2003.

[5] D. W. Albrecht, I. Zukerman, and A. E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *Modelling, User-Adaptation and Interaction*, 8(1-2):5–47, 1998.

[6] D. W. Albrecht, I. Zukerman, and A. E. Nicholson. Pre-sending documents on the www: A comparative study. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1274–1279, 1999.

[7] J. Alexandersson. Plan recognition in verbmobil. In *Proceedings of the IJCAI-95 Workshop. The Next Generation of Plan Recognition Systems: Challenges for and Insight from Related Areas of AI*, pages 2–7, 1995.

[8] J. Allen and C. Perrault. Analyzing intention in utterances. *Artificial Intelligence*, 15:143–178, 1980.

[9] J. Allen. Recognizing intentions from natural language utterances. In M. Brady, editor, *Computational Models of Discourse*. M.I.T. Press, Cambridge, Massachusetts, 1982.

[10] J. R. Anderson. *Learning and Memory: An Integrated Approach*. Wiley, New York, 2 edition, 2000.

[11] P. Angelov and D. Filev. Flexible models with evolving structure. *International Journal Intelligent Systems*, 19(4):327–340, 2004.

[12] P. Angelov and D. Filev. Simpl_ets: a simplified method for learning evolving takagi-sugeno fuzzy models. In *Proceedings of the 14th IEEE International Conference on Fuzzy Systems (FUZZ-05)*, pages 1068–1073, 2005.

[13] P. Angelov and X. Zhou. Evolving fuzzy rule-based classifiers from data streams. *IEEE Transactions on Fuzzy Systems, Special issue on Evolving Fuzzy Systems*, 16(6):1462–1475, 2008.

[14] P. Angelov, X. Zhou, and F. Klawonn. Evolving fuzzy rule-based classifiers. In *Proceedings of the IEEE Symposium on Computational Intelligence in Image and Signal Processing (CIISP-07)*, pages 220–225, 2007.

[15] P. Angelov. *Evolving Rule-based Models: A Tool for Design of Flexible Adaptive Systems*. Springer-Verlag, New York, 2002.

[16] K. Asai, S. Hayamizu, and K. Handa. Prediction of protein secondary structure by the hidden markov model. *Computer Applications in the Biosciences*, 9(2):141–146, 1993.

[17] D. Avrahami-Zilberbrand. Symbolic behavior recognition. Master's thesis, Bar Ilan University, 2004.

[18] J. Bajo, J. M. Corchado, and S. Rodríguez. GR-MAS: Multi-Agent System for Geriatric Residences. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-08)*, pages 875–876, 2008.

[19] P. Bakker and Y. Kuniyoshi. Robot See, Robot Do: An overview of robot imitation. In *Proceedings of AISB-96 Workshop on Learning in Robots and Animals*, pages 3–11, 1996.

[20] T. R. Balch. *Behavioral diversity in learning robot teams*. PhD thesis, Georgia Institute of Technology, 1998.

[21] P. Baldi, S. Brunak, Y. Chauvin, and A. Krogh. Naturally occurring nucleosome positioning signals in human exons and introns. *Journal of Molecular Biology*, 263(4):503–510, 1996.

[22] C. Basu, H. Hirsh, and W. W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the National Conference on Artificial Intelligence(AAAI-98)*, pages 714–720, 1998.

[23] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

[24] Y. Bengio. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162, 1999.

[25] M. Bicego, V. Murino, and M. A.T. Figueiredo. Similarity-based classification of sequences using hidden markov models. *Pattern Recognition*, 37(12):2281 – 2291, 2004.

[26] D. Billsus and M. J. Pazzani. A personal news agent that talks, learns and explains. In *Proceedings of the third annual conference on Autonomous Agents (AGENTS-99)*, pages 268–275, 1999.

[27] L. Breiman, J. Friedman, C. J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Chapman & Hall/CRC, January 1984.

[28] H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov models. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

[29] H. Bui. A general model for online probabilistic plan recognition. In *Proceedings of the International joint conference on Artificial Intelligence (IJCAI-03)*, pages 1309–1318, 2003.

[30] S. Buttinger, M. Diedrich, L. Hennig, A. Hoenemann, P. Huegelmeyer, A. Nie, A. Pegam, C. Rogowski, C. Rollinger, T. Steffens, and W. Teiken. The Dirty Dozen Team and Coach Description. In *Proceedings of the Robot Soccer World Cup V (RoboCup-01)*, pages 543–546, 2002.

[31] S. Carberry. Incorporating default inferences into plan recognition. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-90)*, pages 471–478, 1990.

[32] S. Carberry. *Plan Recognition in Natural Language Dialogue*. MIT Press, Cambridge, MA, USA, 1990.

[33] S. Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48, 2001.

[34] J. Carbo, A. Orfila, and A. Ribagorda. Adaptive agents applied to intrusion detection. In *Proceedings of the International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS-03)*, pages 445–453, 2003.

[35] J. Carbo, A. Orfila, and A. Ribagorda. Autonomous decision on intrusion detection with trained BDI agents. *Computer Communications*, 31(9):1803–1813, 2008.

[36] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-96)*, pages 120–125, 1996.

[37] D. Carmel and S. Markovitch. Learning models of intelligent agents. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 62–67, 1996.

[38] G. A. Carpenter and S. Grossberg. Adaptive resonance theory. *The handbook of brain theory and neural networks*, pages 79–82, 2003.

[39] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen. Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698–713, 1992.

[40] C. Castelfranchi. Guarantees for autonomy in cognitive agent architecture. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 56–70, 1995.

[41] K. M. A. Chai, H. L. Chieu, and H. T. Ng. Bayesian online classifiers for text classification and filtering. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-02)*, pages 97–104, 2002.

[42] E. Charniak and R. P. Goldman. A bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.

[43] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1985.

[44] E. Charniak. Bayesian networks without tears. *AI Magazine*, pages 50–63, 1991.

[45] M. Chen, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanaskis, K. Kostiadis, J. Kummeneje, I. Noda, O. Obst, P. Riley, T. Steffens, Y. Wang, and X. Yin, 2001. Users Manual RoboCup Soccer Server version 7.07.

[46] C. L. Chiang. Statistical methods of analysis. *World Scientific*, 2003.

[47] P. R. Cohen, C. R. Perrault, and J. F. Allen. Beyond question answering. In W. G. Lehnert and M. H. Ringle, editors, *Strategies for Natural Language Processing*, pages 245–274. 1982.

[48] D. Cook and M. Schmitter-Edgecombe. Assessing the quality of activities in a smart environment. *Methods of Information in Medicine*, 48(5):480–485, 2009.

[49] D. Cook, M. Schmitter-Edgecombe, A. Crandall, C. Sanders, and B. Thomas. Collecting and disseminating smart home sensor data in the casas project. In *Proceedings of CHI09 Workshop on Developing Shared Home Behavior Datasets to Advance HCI and Ubiquitous Computing Research*, 2009.

[50] J. M. Corchado and B. Lees. A hybrid case-based model for forecasting. *Applied Artificial Intelligence*, 15(2):105–127, 2001.

[51] J. M. Corchado and J. M. Molina. *Introducción a la Teoría de Agentes y Sistemas Multiagente*. Edite Publicaciones Científicas, 2002.

[52] S. E. Coull, J. W. Branch, B. K. Szymanski, and E. Breimer. Intrusion detection: A bioinformatics approach. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC-03)*, pages 24–33, 2003.

[53] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[54] R. López de Mántaras, D. McSherry, D. Bridge, D. Leake, B. Smyth, S. Craw, B. Faltings, M. Maher, M. Cox, K. Forbus, M. Keane, and I. Watson. Retrieval, Reuse, Revision, and Retention in CBR. *The Knowledge Engineering Review*, 20(3):215–240, 2006.

[55] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[56] M. Devaney and A. Ram. Needles in a haystack: Plan recognition in large spatial domains involving multiple agents. In *Proceedings of the National Conference on Artificial Intelligence(AAAI-98)*, pages 942–947, 1998.

[57] M. d´Inverno and M. Luck. *Understanding agent systems*. Springer series on agent technology. 2001.

[58] P. Domingos and G. Hulten. Catching up with the data: Research issues in mining data streams. In *Proceedings of the Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2001.

[59] G. Dong and J. Pei. *Sequence data mining*. Springer, 2007.

[60] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons Inc, 1973.

[61] R. O. Duda, P. E. Hart, and D.G. Stork. *Pattern Classification*. Wiley-Interscience, 2000.

[62] T. Duong, H. H. Bui, D. Phung, and S. Venkatesh. Activity recognition and abnormality detection with the switching hidden semi-markov models. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR-05)*, 2005.

[63] K. Erol, J. A. Hendler, and D. S. Nau. UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS-94)*, pages 249–254, 1994.

[64] O. Etzioni. The world wide web: quagmire or gold mine? *Communications of the ACM*, 39:65–68, 1996.

[65] R. Fathzadeh, V. Mokhtari, and M. R. Kangavari. Opponent provocation and behavior classification: A machine learning approach. In *Proceedings of the Robot Soccer World Cup XI (RoboCup-07)*, pages 540–547, 2008.

[66] J. Ferber. *Multi-Agent Systems*. Addison Wesley, 1999.

[67] F. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. Riquelme. Incremental rule learning based on example nearness from numerical data streams. In *Proceedings of the ACM symposium on Applied computing 2005*, pages 568–572, 2005.

[68] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, volume 1193, 1996.

[69] P. Frasconi, M. Gori, M. Maggini, and G. Soda. Unified integration of explicit knowledge and learning by example in recurrent networks. *IEEE Transactions on Knowledge and Data Engineering*, 7(2):340–346, 1995.

[70] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[71] Y. Gal and A. Pfeffer. Networks of influence diagrams: A formalism for representing agents beliefs and decision-making processes. *Journal of Artificial Intelligence Research*, 33:109–147, 2008.

[72] J. García, J. Carbó, and J. M. Molina. Agent-based coordination of cameras. *International Journal of Computer Science & Applications*, 2(1):33–37, 2005.

[73] C. Geib and R. Goldman. Plan recognition in intrusion detection systems. In *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX)*, 2001.

[74] C. W. Geib and R. P. Goldman. Probabilistic plan recognition for hostile agents. In *Proceedings of the 14th International Florida Artificial Intelligence Research Society Conference (FLAIRS-01)*, pages 580–584, 2001.

[75] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 147, 1994.

[76] A. Ghosh, M. Biehl, and B. Hammer. Performance analysis of lvq algorithms: a statistical physics approach. *Neural Networks*, 19(6):817–829, 2006.

[77] C. Lee Giles, B. G. Horne, and T. Lin. Learning a class of large finite state machines with a recurrent neural network. Technical report, University of Maryland at College Park, College Park, MD, USA, 1994.

[78] D. Godoy and A. Amandi. User profiling for web page filtering. *IEEE Internet Computing*, 9(4):56–64, 2005.

[79] E. Gold. Complexity of automaton identification for given data. *Information and Control*, 37:302–320, 1978.

[80] B. A. Goodman and D. J. Litman. On the interaction between plan recognition and intelligent interfaces. *User Modeling and User-Adapted Interaction*, 2(1-2):55–82, 1992.

[81] S. Greenberg and I. H. Witten. Directing the user interface: how people use command-based systems. In *Proceedings of the 3rd IFAC Conference on Man-Machine Systems*, 1988.

[82] S. Greenberg and I. H. Witten. How users repeat their actions on computers: principles for design of history mechanisms. In *Proceedings of the Canadian Information Processing Society Edmonton Conference*, pages 171–178, 1988.

[83] S. Greenberg. Tool use, reuse, and organization in command-driven interfaces. Master's thesis, Department of Computer Science, University of Calgary, Alberta, 1988.

[84] S. Greenberg. Saul greenberg web page, 2009. http://pages.cpsc.ucalgary.ca/~saul/wiki/pmwiki.php.

[85] J. T. Hackos and J. C. Redish. *User and Task Analysis for Interface Design*. Wiley, 1998.

[86] S. M. Haller and S. C. Shapiro. IDP — an interactive discourse planner. In *Trends in Natural Language Generation: An Artificial Intelligence Perspective , Lecture Notes in Artificial Intelligence 1036*, pages 144–167. Springer-Verlag, 1996.

[87] K. Han and M. Veloso. Automated Robot Behavior Recognition Applied to Robotic Soccer. In *Robotics Research: the 9th International Symposium*, pages 199–204. 2000.

[88] J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[89] Y. Horman and G. Kaminka. Removing statistical biases in unsupervised sequence learning. In *Proceedings of Intelligent Data Analysis (IDA-05)*, pages 157–167, 2005.

[90] A. E. Howe and P. R. Cohen. Understanding planner behavior. *Artificial Intelligence*, 76(1-2):125–166, 1995.

[91] Z. Huang, Y. Yang, , and X. Chen. An approach to plan recognition and retrieval for multi-agent systems. In *Proceedings of the 1st RoboCup Australian Open 2003 (AORC-03)*, 2003.

[92] K. Huff and V. R. Lesser. Knowledge-based command understanding. an example for the software development environment. Technical report, Computer and Information Sciences University of Massachusetts at Amherst, Amherst, Mass, 1982.

[93] M. N. Huhns and M. P. Singh. *Readings in agents*. San Francisco, CA, USA, 1998.

[94] J. A. Iglesias, A. Ledezma, and A. Sanchis. CAOS Coach Team - Participant in the RoboCup Coach 2006, 2006. http://www.caos.inf.uc3m.es/caoscoachteam/index.htm.

[95] J. A. Iglesias, J. A. Fernández, I. R. Villena, A. Ledezma, and A. Sanchis. The winning advantage: Using opponent models in robot soccer. In *Proceedings of the The 10th International Conference on Intelligent Data Engineering and Automated Learning (IDEAL-09)*, pages 485–493, 2009.

[96] S. S. Intille and A. F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the National Conference on Artificial Intelligence(AAAI-99)*, pages 518–525, 1999.

[97] S. Iravanian, M. M. Mashadi, E. Rafiee, and A. A. Lohrasbi. Mehr coach 2006: Offline learning and online pattern detection. In *Proceedings CD RoboCup 2006*, June 2006.

[98] A. J., B. Grosmann-Hutter, L. March, and R. Rummer. Creating an empirical basis for adaptation decisions. In *Intelligent User Interfaces*, pages 149–156, 2000.

[99] P. J. Jansen. Problematic positions and speculative play. *Computers, Chess and cognition*, pages 169–182, 1990.

[100] P. J. Jansen. Kqkr: Speculatively thwarting a human opponent. *International Computer Chess Association Journal*, 16(1):3–17, 1993.

[101] W. Joy. *An introduction to the C shell. Unix Programmer´s Manual*. University of California, Berkley, California, 1980.

[102] G. Kaminka and M. H. Bowling. Towards robust teams with many agents. In *Proceedings of Autonomous Agents and Multi Agent Systems (AAMAS-02)*, pages 729–736, 2002.

[103] G. Kaminka and M. Tambe. Robust agent teams via socially-attentive monitoring. *Journal of Artificial Intelligence Research*, 12:105–147, 2000.

[104] G. Kaminka, M. Fidanboylu, A. Chang, and M. Veloso. Learning the sequential coordinated behavior of teams from observations. In *Proceedings of the Robot Soccer World Cup VI (RoboCup-02)*, June 2002.

[105] G. Kaminka, D. V. Pynadath, and M. Tambe. Monitoring teams by overhearing: A multi-agent plan recognition approach. *Journal of Artificial Intelligence Research*, 17, 2002.

[106] T. Kanungo. Umdhmm: A hidden markov model toolkit. In *Proceedings of the Extended Finite State Models of Language*. Cambridge University Press, 1999. [Online] Available: http://www.cfar.umd.edu/ kanungo/software/software.html.

[107] N. Kasabov and Q. Song. DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction. *IEEE Transactions on Fuzzy Systems*, 10(2):144–154, 2002.

[108] H. A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, pages 32–37, 1986.

[109] J. Kay and E. McCreath. Automatic induction of rules for email classification. In *Proceedings of the 2001 Workshop on Machine Learning for User Modeling (ML4UM-01)*, pages 59–66, 2001.

[110] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada. The RoboCup Synthetic Agent Challenge 97. In *Proceedings of 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 24–29, 1997.

[111] R. Klinkenberg and T. Joachims. Detection concept drift with support vector machines. In *Proceedings of the International Conference on Machine Learning (ICML-00)*, pages 487–494, 2000.

[112] D. Knuth. *Searching and Sorting*. Addison-Wesley, 1975.

[113] T. Kohonen. The self-organizing map. *Neurocomputing*, 21(1-3):1–6, 1998.

[114] R. E. Korf. Generalized game trees. In *Proceedings of 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 328–333, 1989.

[115] R. Kosala and H. Blockeel. Web mining research: a survey. *SIGKDD Explorations*, 2(1):1–15, 2000.

[116] G. Kuhlmann, P. Stone, and J. Lallinger. The champion UT Austin Villa 2003 Simulator Online Coach Team. In *Proceedings of the Robot Soccer World Cup VII (RoboCup-03)*.

[117] G. Kuhlmann, P. Stone, and J. Lallinger. The UT Austin Villa 2003 Champion Simulator Coach: A Machine Learning Approach. In *Proceedings of the Robot Soccer World Cup VIII (RoboCup-04)*, pages 636–644, 2004.

[118] G. Kuhlmann, B. Knox, and P. Stone. The ut austin villa 2006 simulator coach. In *Proceedings CD RoboCup 2006*, June 2006.

[119] G. Kuhlmann, W. B. Knox, and P. Stone. Know thine enemy: A champion RoboCup coach agent. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, pages 1463–68, 2006.

[120] P. Laird and R. Saul. Discrete sequence prediction and its applications. *Machine Learning*, 15(1):43–68, 1994.

[121] T. Lane and C. E. Brodley. Sequence matching and learning in anomaly detection for computer security. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-97). Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 43–49, 1997.

[122] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, 1999.

[123] T. Lane and C. E. Brodley. An empirical study of two approaches to sequence learning for anomaly detection. *Machine Learning*, 51(1):73–107, 2003.

[124] T. Lane. Hidden markov models for human/computer interface modeling. In *Proceedings of the IJCAI-99 Workshop on Learning About Users*, pages 35–44, 1999.

[125] T. Lane. Uci machine learning repository, 2009. http://archive.ics.uci.edu/ml/datasets/UNIX+User+Data.

[126] A. Ledezma, R. Aler, A. Sanchis, and D. Borrajo. Ombo: An opponent modeling approach. *AI Communications*, 22(1):21–35, 2009.

[127] K. Lerman and A. Galstyan. Automatically modeling group behavior of simple agents. In *Proceedings of the International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS-04)*, pages 49–55, 2004.

[128] N. Lesh, C. Rich, and C. L. Sidner. Using plan recognition in human-computer collaboration. In *Proceedings of the 7th international conference on User modeling (UM-99)*, pages 23–32, 1999.

[129] L. Liao, D. Fox, and H. A. Kautz. Location-based activity recognition using relational markov networks. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 773–778, 2005.

[130] H. Lida, J. W. H. M. Uiterwijk, and I.S. Herschberg. Potential applications of opponent-model search. In *Proceedings of the 7th Conference on Advances in Computer Chess*, pages 201–208, 1993.

[131] D. Loewenstern, H. Hirsh, P. Yianilos, and M. Noordewier. DNA sequence classification using compression-based induction. Technical Report 95–04, DIMACS, 1995.

[132] m. Schonlau. Matt schonlau web page - masquerading user data, 2009. http://www.schonlau.net/intrusion.html.

[133] Q. Ma, J. T. Wang, D. Shasha, and C. H. Wu. DNA sequence classification via an expectation maximization algorithm and neural networks: a case study. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 31(4):468–475, 2001.

[134] A. A. Macedo, K. N. Truong, J. A. Camacho-Guerrero'e, and M. da Graça Pimentel. Automatically sharing web experiences through a hyperdocument recommender system. In *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia (HYPERTEXT-03)*, pages 48–56, 2003.

[135] A. Mas. *Agentes Software y sistemas multi-agente. Conceptos ,arquitecturas y aplicaciones*. 2004.

[136] H. McGilton and R. Morgan. *Introducing the Unix System*. 1983.

[137] A. Mihailidis, J. C. Barbenel, and G. R. Fernie. The efficacy of an intelligent cognitive orthosis to facilitate handwashing by persons with moderate-to-severe dementia. *Neuropsychological Rehabilitation*, 14(1):135–171, 2004.

[138] N. J. Mulcahy and J. Call. Apes save tools for future use. *Science*, 312(5776):1038–1040, 2006.

[139] B. A. Myers. A brief history of human-computer interaction technology. *ACM Interactions*, 5(2):44–54, 1998.

[140] K. Nakai. UCI repository of machine learning databases - ecoli data set, 1998. http://archive.ics.uci.edu/ml/datasets/Ecoli.

[141] J. V. Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[142] A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–127, 1982.

[143] M. N. Nicolescu and M. J. Matarić. A hierarchical architecture for behavior-based robots. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS-02)*, pages 227–233, 2002.

[144] P. Nixon, W. Wagealla, C. English, and S. Terzis. *Privacy, Security, and Trust Issues in Smart Environments*, chapter Smart Environments: Technology, Protocols and Applications, pages 220–240. 2004.

[145] I. Noda, H. Matsubara, K. Hiraki, and I. Frank. Soccer server. a tool for research on multiagent systems. *Applied Artificial Intelligence*, 12(2-3):233–250, 1998.

[146] J. Orwant. Heterogeneous learning in the doppelgänger user modeling system. *User Modeling User-Adaptatation and Interaction*, 4(2):107–130, 1995.

[147] S. Pang, S. Ozawa, and N. Kasabov. Incremental linear discriminant analysis for classification of data streams. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 35(5):905–914, 2005.

[148] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of markov decision procedures. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[149] M. Philipose, K. P. Fishkin, M. Perkowitz, D. J. Patterson, D. Fox, H. Kautz, and D. Hahnel. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 3(4):50–57, 2004.

[150] M. Piccardi and O. Perez. Hidden markov models with kernel density estimation of emission probabilities and their use in activity recognition. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 0:1–8, 2007.

[151] O. Pérez, M. Piccardi, J. García, M. A. Patricio, and J. M. Molina. Comparison between genetic algorithms and the baum-welch algorithm in learning hmms for human activity classification. In *Proceedings of the EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog*, pages 399–406, 2007.

[152] CASAS Project. CASAS smart home project, 2009. http://ailab.eecs.wsu.edu/casas/.

[153] D. V. Pynadath and M. P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 507–514, 2000.

[154] J.R. Quinlan. Data mining tools see5 and c5.0, 2003 [online]. Available: http://www.rulequest.com/see5-info.html.

[155] L. Rabiner and B. Juang. *Fundamentals of speech recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[156] B. Reisberg, S. Finkel, J. Overall, N. Schmidt-Gollas, S. Kanowski, H. Lehfeld, F. Hulla, S. G. Sclan, H.U. Wilms, K. Heininger, I. Hindmarch,

M. Stemmler, L. Poon, A. Kluger, C. Cooler, M. Bergener, and L. Hugonot-Diener. The Alzheimer's disease activities of daily living international scale (ADL-IS). *International Psychogeriatrics*, 13(2):163–81, 2001.

[157] P. Riley and M. Veloso. On behavior classification in adversarial environments. In *Distributed Autonomous Robotic Systems 4*, pages 371–380. 2000.

[158] P. Riley and M. Veloso. Planning for distributed execution through use of probabilistic opponent models. In *Proceedings of the 6th International Conference on AI Planning and Scheduling (AIPS-02)*, 2002.

[159] P. Riley. *Coaching: Learning and Using Environment and Agent Models for Advice*. PhD thesis, Computer Science Dept., Carnegie Mellon University, 2005.

[160] D. Sánchez, M. Tentori, and J. Favela. Activity recognition for the smart hospital. *IEEE Intelligent Systems*, 23(2):50–57, 2008.

[161] S. Saria and S. Mahadevan. Probabilistic plan recognition in multiagent systems. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, pages 287–296, 2004.

[162] C. F. Schmidt, N. S. Sridharan, and J. L. Goodson. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence*, 11(1-2):45–83, 1978.

[163] M. Schonlau, W. Dumouchel, W. H. Ju, A. F. Karr, and Theus. Computer intrusion: Detecting masquerades. In *Statistical Science*, volume 16, pages 58–74, 2001.

[164] R. B. Segal and J. O. Kephart. Mailcat: an intelligent assistant for organizing e-mail. In *Proceedings of the 3rd annual conference on Autonomous Agents (AGENTS-99)*, pages 276–282, 1999.

[165] C. E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[166] C. L. Sidner and D. J. Israel. Recognizing intended meaning and speakers' plans. In *Proceedings of International Joint Conference on Artificial Intelligence*, pages 203–208, 1981.

[167] D. Singla, D. Cook, and M. Schmitter-Edgecombe. Memory deficits, everyday functioning, and mild cognitive impairment. In *Proceedings of the Annual Rehabilitation Psychology Conference*, 2008.

[168] R. Sison and M. Shimura. Student modeling and machine learning. *International Journal of Artificial Intelligence in Education*, 9:128–158, 1998.

[169] J. Söding. Protein homology detection by hmm-hmm comparison. *Bioinformatics (Oxford, England)*, 21(7):951–960, 2005.

[170] M. Spiliopoulou and L. C. Faulstich. Wum: A web utilization miner. In *Proceedings of International Conference on Extending Database Technology (EDBT-04)*, pages 109–115, 1998.

[171] T. Steffens. Adapting similarity measures to agent types in opponent modelling, 2004.

[172] P. Stone, P. Riley, and M. Veloso. Defining and using ideal teammate and opponent models. In *Proceedings of the 12th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 1040–1045, 2000.

[173] K. P. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.

[174] t. Steffens. Feature-based declarative opponent-modelling in multi-agent systems. Master's thesis, Institute of Cognitive Science Osnabrück, 2002.

[175] T. Takagi and M. Sugeno. Fuzzy identification of systems and its application to modeling and control. *IEEE Transactions on System, Man, and Cybernetics*, 15(1):116–132, 1985.

[176] T. Takahashi. Kasugabito III. In *Proceedings of the Robot Soccer World Cup III (RoboCup-1999)*, pages 592–595, 2000.

[177] M. Tambe and P. S. Rosenbloom. RESC: An approach for dynamic, real-time agent tracking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, 1995.

[178] M. Tambe. Tracking dynamic team activity. In *Proceedings of the National Conference on Artificial Intelligence(AAAI-96)*, pages 80–87, 1996.

[179] T. S. Tan, F. Dillon, E. Hadzic, and E. Chang. In SEQUEST: mining frequent subsequences using DMA Strips, editor, *Proceedings of the 7th International Conference on Data Mining and Information Engineering*, pages 35–328, 2006.

[180] M. F. Umer and M. Khiyal. Classification of textual documents using learning vector quantization. *Information Technology Journal*, (6):154–159, 2007.

[181] D. L. Vail and M. Veloso. Feature selection for activity recognition in multi-robot domains. In *AAAI*, pages 1415–1420, 2008.

[182] P. Valckenaers, J. Sauter, C. Sierra, and J. Rodriguez-Aguilar. Applications and environments for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):61–85, 2007.

[183] J. T. L. Wang, Q. Ma, D. Shasha, and C. H. Wu. New techniques for extracting features from protein sequences. *IBM Systems Journal*, 40(2):426–436, 2001.

[184] The RoboCup Coach Competition Web Page.

[185] G. I. Webb and M. Kuzmycz. Feature based modelling: A methodology for producing coherent, consistent, dynamically changing models of agent's competencies. *User Modeling and User-Adapted Interaction*, 5(2):117–150, 1996.

[186] G. I. Webb, M. J. Pazzani, and D. Billsus. Machine learning for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2):19–29, 2001.

[187] G. M. Weiss and H. Hirsh. Learning to predict rare events in event sequences. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD-98)*, pages 359–363, 1998.

[188] Gerhard Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. In *Machine Learning*, pages 69–101, 1996.

[189] D. Willingham, M. Nissen, and P. Bullemer. On the development of procedural knowledge. *Experimental Psychology: Learning, Memory, and Cognition*, 15:1047–1060, 1989.

[190] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

[191] M. Wooldridge. Agent-based software engineering. *IEE Proceedings Software Engineering*, 144(1):26–37, 1997.

[192] C. R. Wren and E. Munguia. Toward scalable activity recognition for sensor networks. In *Proceedings of the 2nd International Workshop on Location- and Context-Awareness (LoCA-06)*, pages 168–185, 2006.

[193] J. Wu, P. Zhang, Z. Xiong, and H. Sheng. Mining personalization interest and navigation patterns on portal. In *Proceedings of the 11th Pacific-Asia Knowledge Discovery and Data Mining conference (PAKDD-07)*, pages 948–955, 2007.

[194] Y. Xu, S. Furao, O. Hasegawa, and J. Zhao. An online incremental learning vector quantization. In *Proceedings of the Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining (PAKDD-09)*, pages 1046–1053, 2009.

[195] J. Yang and W. Wang. Cluseq: Efficient and effective sequence clustering. In *Proceedings of the International Conference on Data Engineering (ICDE-03)*, pages 101–112. IEEE Press, 2003.

[196] J. Yin, Q. Yang, D. Shen, and Z. Li. Activity recognition via user-trace segmentation. *ACM Transactions on Sensor Networks*, 4(4):1–34, 2008.

[197] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[198] I. Zukerman and D. W. Albrecht. Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2):5–18, 2001.

Apéndice A

Competición *RoboCup Coach 2006*

En este apéndice se detallan varios aspectos de la competición *RoboCup Coach 2006* (que se denomina como *Coach 2006*) celebrada del 14 al 18 de Junio del año 2006 en Bremen, Alemania.

En el primer apartado se describen los diferentes patrones utilizados en la competición. En el segundo apartado se describen los patrones de la fase de *Detección online*. El tercer apartado analiza el tipo de acciones consideradas en los patrones y si éstas son modeladas correctamente por M-COMP. Los tres equipos más relevantes de la competición se muestran en el cuarto apartado. Por último, se detallan las puntuaciones de cada equipo y los resultados finales de cada ronda de la competición.

A.1. Configuración de la Competición - *Modelado OffLine*

En este apartado se describen todos los patrones de comportamiento utilizados en la primera parte las dos primeras rondas de la competición.

A.1.1. Patrones de comportamiento - Ronda 1:

Esta primera ronda considera 17 comportamientos diferentes que son nombrados con números del 0 al 16 tal y como se muestra a continuación.

- *patrón_00:* Los jugadores 3 y 4 siempre pasan la pelota al recibirla.
- *patrón_01:* Los jugadores 7, 9, 10 y 11 siempre pasan la pelota al recibirla.
- *patrón_02:* Los jugadores 9, 10 y 11 siempre pasan a un determinado punto del campo obtenido en función de la posición de la pelota.

- *patrón_03:* Los jugadores 2, 3, 4 y 5 siempre regatean hasta un punto determinado.
- *patrón_04:* Los jugadores 6, 7 y 8 siempre pasan a un determinado punto del campo obtenido en función de la posición de la pelota.
- *patrón_05:* Los jugadores 6 y 9 siempre tratan de interceptar la pelota.
- *patrón_06:* El portero siempre se moverá en su área en función de cómo se mueva la pelota en el campo.
- *patrón_07:* Cuando los jugadores 2 y 5 reciben la pelota, la mantienen en el mismo sitio el mayor tiempo posible.
- *patrón_08:* Dependiendo de la posición de la pelota, el jugador 8 regateará a una u otra parte del campo o pasará a cualquier otro jugador.
- *patrón_09:* Este comportamiento está relacionado con la formación del equipo. Todos los jugadores tienen una posición base determinada.
- *patrón_10:* Los jugadores que se encuentren en la mitad del campo del propio equipo deberán pasar al jugador 5 y éste driblar hasta el área del equipo oponente. Cuando la pelota se encuentre en este área, los jugadores 10 y 11 tratarán siempre de disparar a puerta.
- *patrón_11:* El portero siempre pasará al jugador 5 y éste al jugador 3. Cuando el jugador 3 reciba la pelota, éste la sacará fuera del campo.
- *patrón_12:* Los jugadores 2 y 5 siempre pasarán a un punto determinado del campo.
- *patrón_13:* Cuando la pelota se encuentre en el medio campo del oponente, los jugadores 9, 10 y 11 se posicionarán juntos.
- *patrón_14:* El jugador 3 siempre regatea hasta la posición en la que se encuentra el jugador 5. El jugador 5 regatea hasta la posición del jugador 9 y éste hasta la posición en la que se encuentra el jugador 3.
- *patrón_15:* El portero siempre pasará al jugador 2 y éste pasará al jugador 10. El jugador 10 siempre lanzará la pelota a un punto determinado del campo.
- *patrón_16:* El portero siempre pasará a los jugadores 3 o 4, los cuales regatearán hasta un determinado punto del medio campo del equipo contrario.

A.1.2. Patrones de comportamiento - Ronda 2:

Para esta segunda ronda, se utilizaron 16 patrones de comportamientos que se nombran con letras de la A a la P.

- *patrón_A:* Los jugadores 5, 6 y 7 siempre pasarán a un determinado punto del campo.
- *patrón_B:* Cuando los jugadores 2, 3 y 4 reciban la pelota, siempre pasarán a un punto muy cercano al que se encuentran.
- *patrón_C:* Los jugadores 3 y 4 siempre pasarán a un punto del campo.
- *patrón_D:* Los jugadores 2 y 5 tratarán de sacar la pelota de su campo.
- *patrón_E:* Los jugadores 7 y 8 siempre regatearán hasta un determinado punto del campo obtenido en función de la posición de la pelota.
- *patrón_F:* El portero se aproximará a la pelota siempre que ésta se encuentre en su medio campo.
- *patrón_G:* Cuando la pelota se encuentra en el medio campo del oponente, el jugador 4 se distanciará de la pelota, sino, se acercará a ella.
- *patrón_H:* Los jugadores 3 y 4 siempre pasarán al portero.
- *patrón_I:* Cuando el jugador 6 reciba la pelota, éste regateará hasta el área del equipo oponente.
- *patrón_J:* Los jugadores que se encuentren en la mitad del campo del propio equipo deberán pasar a los jugadores 10 y 11 que tratarán siempre de disparar a puerta.
- *patrón_K:* El jugador 10 siempre pasará al jugador 9. El jugador 9 siempre pasará al jugador 10 y éste regateará hasta un punto próximo a la portería del oponente.
- *patrón_L:* El portero siempre pasará al jugador 2 y éste al jugador 3.
- *patrón_M:* El jugador 10 regatea hasta el área del equipo oponente y allí disparará a puerta.
- *patrón_N:* Los jugadores 9, 10 y 11 siempre regatearán hasta un determinado punto del campo obtenido en función de la posición de la pelota.
- *patrón_O:* El jugador 10 siempre pasará a un punto próximo al jugador 2.

- *patrón_P:* Los jugadores 2, 3 y 5 siempre estarán situados en la mitad del campo del equipo oponente.

A.2. Configuración de la Competición - *Detección OnLine*

En la fase de *Detección online*, es posible que algunos patrones no se ejecuten o lo hagan muy pocas veces. La Tabla A.1 muestra los jugadores involucrados en un determinado patrón y si éstos han realizado o no la acción definida en dicho patrón. Además, si el jugador no ha realizado la acción correspondiente, se diferencia si ha sido por la intervención de otro jugador o porque ha hecho otra acción diferente a la indicada. En este caso, se muestran los patrones de la iteración 1 de la ronda 1.

La Tabla A.2 analiza del mismo modo los patrones utilizados en las dos últimas iteraciones de la ronda 1. Por último, la Tabla A.3 muestra este mismo análisis en la segunda ronda. En este caso, la iteración 1 no incluye el *patrón_J* ni la iteración 2 el *patrón_P* porque éstos hacen referencia al posicionamiento de los jugadores.

Tabla A.1: Ejecución de los patrones en Iteración 1 - Ronda 1 de competición *Coach 2006*

				El jugador no ejecuta el comportamiento	
Nombre del patrón	**Jugadores Involucrados**	Nº de veces es posible realizar el comportam. en el partido	Comportam. ejecutado correctamente (%)	Otro jugador interviene en la acción (%)	Realiza un comportam. diferente. (%)
patrón_00	Jugador 4	18	66.6	33.3	0
	Jugador 11	3	33.3	66.6	0
patrón_04	Jugador 6	3	100	0	0
	Jugador 7	4	100	0	0
	Jugador 8	1	100	0	0
patrón_14	Jugador 3	8	25	50	25
	Jugador 5	7	14.3	57.1	28.6
	Jugador 9	1	100	0	0
patrón_15	Jugador 1	30	20	70	10
	Jugador 2	13	38.5	53.8	7.7
	Jugador 10	8	100	0	0

Tabla A.2: Ejecución de los patrones en Iteraciones 2 y 3 - Ronda 1 de la competición *Coach 2006*

				El jugador no ejecuta el comportamiento	
Nombre del patrón	**Jugadores Involucrados**	Nº de veces es posible realizar el comportam. en el partido	Comportam. ejecutado correctamente (%)	Otro jugador interviene en la acción (%)	Realiza un comportam. diferente. (%)
RONDA 1 - ITERACIÓN 2					
patrón_01	Jugador 7	0	0	0	0
	Jugador 9	5	80	20	0
	Jugador 10	1	100	0	0
	Jugador 11	4	100	0	0
patrón_07	Jugador 2	7	71.4	28.6	10
	Jugador 5	8	62.5	12.5	25
patrón_08	Jugador 8	3	66.6	33.3	0
patrón_13	Jugador 11	3	100	0	0
	Jugador 9	4	75	25	0
	Jugador 10	2	100	0	0
patrón_16	Jugador 1	22	59.1	36.4	4.5
	Jugador 3	26	61.5	34.6	3.9
	Jugador 4	15	53.4	33.3	13.3
RONDA 1 - ITERACIÓN 3					
patrón_00	Jugador 3	19	57.9	31.6	10.5
	Jugador 4	23	69.6	30.4	0
patrón_02	Jugador 9	8	100	0	0
	Jugador 10	0	0	0	0
	Jugador 11	15	100	0	0
patrón_04	Jugador 6	14	78.6	7.1	14.3
	Jugador 7	0	0	0	0
	Jugador 8	9	77.8	11.1	11.1
patrón_06	Jugador 1	12	75	25	0
patrón_12	Jugador 2	0	0	0	0
	Jugador 5	8	62.5	25	12.5

Tabla A.3: Ejecución de los patrones en la Ronda 2 - Iteraciones 1, 2 y 3 de la competición *Coach 2006*.

				El jugador no ejecuta el comportamiento	
Nombre del patrón	**Jugadores Involucrados**	Nº de veces es posible realizar el comportam. en el partido	Comportam. ejecutado correctamente (%)	Otro jugador interviene en la acción (%)	Realiza un comportam. diferente. (%)
RONDA 2 - ITERACIÓN 1					
patrón_A	Jugador 5	14	57.1	28.6	14.3
	Jugador 6	3	66.6	0	33.3
	Jugador 7	16	43.8	18.8	37.4
patrón_F	Jugador 1	7	85.7	14.3	0
patrón_H	Jugador 3	18	33.3	33.3	33.3
	Jugador 4	0	0	0	0
patrón_O	Jugador 10	6	33.3	33.3	33.3
RONDA 2 - ITERACIÓN 2					
patrón_E	Jugador 8	15	66.6	20	13.3
	Jugador 7	3	66.6	33.3	0
patrón_I	Jugador 6	5	20	40	40
patrón_K	Jugador 10	8	50	50	0
	Jugador 9	3	66.6	33.3	0
	Jugador 11	17	41.2	47.1	11.7
RONDA 2 - ITERACIÓN 3					
patrón_B	Jugador 2	8	62.5	37.5	0
	Jugador 3	17	82.3	11.8	5.9
	Jugador 5	14	71.4	28.6	0
patrón_E	Jugador 8	9	77.7	22.3	0
	Jugador 7	8	62.5	37.5	0
patrón_I	Jugador 6	7	42.9	42.9	14.2
patrón_M	Jugador 10	13	61.5	38.5	0

A.3. Análisis de los patrones considerando los eventos futbolísticos definidos en M-COMP

La definición de los eventos futbolísticos en el MCSE de M-COMP, es un factor fundamental para la detección de un patrón. En este caso, los eventos seleccionados sólo hacen referencia a las acciones realizadas cuando los jugadores tienen el balón.

Las Tablas A.4, A.5 y A.6 analizan cada uno de los patrones utilizados en las tres rondas de las dos iteraciones de la competición *Coach 2006* considerando si las acciones que lo forman están o no relacionadas con los eventos futbolísticos definidos.

Estas tablas muestra cómo existen patrones que son reconocidos a pesar de que las acciones no están directamente relacionadas con los eventos futbolísticos propuestos. Esto se debe a que los eventos no reconocidos implican otros eventos que sí pueden serlo. Por ejemplo, en el *patrón_B*, los jugadores 2, 3 y 4 siempre pasarán a un punto muy cercano al que se encuentran. Este evento en concreto no es reconocido como tal, sino como un pase a un determinado jugador. Así, en la iteración 3 de la ronda 2, este patrón es reconocido.

Tabla A.4: Análisis del tipo de acción del patrón - Ronda 1, Iteración 1 de la competición *Coach 2006*

Nombre del patrón	**Jugadores Involucrados**	¿Acciones relacionadas con los eventos? definidos	¿Acción relevante en el modelo creado?	¿Patrón detectado?
patrón_00	Jugador 4	no	no	No
	Jugador 11	no	no	
patrón_04	Jugador 6	no	no	No
	Jugador 7	no	no	
	Jugador 8	no	no	
patrón_14	Jugador 3	si	si	Si
	Jugador 5	si	si	
	Jugador 9	si	si	
patrón_15	Jugador 1	si	si	Si
	Jugador 2	si	si	
	Jugador 10	si	si	

Tabla A.5: Análisis del tipo de acción del patrón - Ronda 1, Iteraciones 2 y 3 de la competición *Coach 2006*

Nombre del patrón	**Jugadores Involucrados**	¿Acciones relacionadas con los eventos? definidos	¿Acción relevante en el modelo creado?	¿Patrón detectado?
RONDA 1 - ITERACIÓN 2				
patrón_01	Jugador 7	no	no	No
	Jugador 9	no	no	
	Jugador 10	no	no	
	Jugador 11	no	no	
patrón_07	Jugador 2	si	no	Si
	Jugador 5	si	no	
patrón_08	Jugador 8	no	no	No
patrón_13	Jugador 11	no	no	No
	Jugador 9	no	no	
	Jugador 10	no	no	
patrón_16	Jugador 1	si	si	Si
	Jugador 3	si	si	
	Jugador 4	si	si	
RONDA 1 - ITERACIÓN 3				
patrón_00	Jugador 3	no	si	Si
	Jugador 4	no	no	
patrón_02	Jugador 9	no	si	Si
	Jugador 10	no	no	
	Jugador 11	no	si	
patrón_04	Jugador 6	no	no	No
	Jugador 7	no	no	
	Jugador 8	no	no	
patrón_06	Jugador 1	no	no	No
patrón_12	Jugador 2	no	no	No
	Jugador 5	no	no	

Tabla A.6: Análisis del tipo de acción del patrón - Ronda 2, Iteraciones 1, 2 y 3 de la competición *Coach 2006*

Nombre del patrón	**Jugadores Involucrados**	¿Acciones relacionadas con los eventos? definidos	¿Acción relevante en el modelo creado?	¿Patrón detectado?
RONDA 2 - ITERACIÓN 1				
patrón_A	Jugador 5	no	no	No
	Jugador 6	no	no	
	Jugador 7	no	no	
patrón_F	Jugador 1	no	si	Si
patrón_H	Jugador 3	si	si	No
	Jugador 4	si	no	
patrón_O	Jugador 10	si	si	No
RONDA 2 - ITERACIÓN 2				
patrón_E	Jugador 8	si	no	No
	Jugador 7	si	si	
patrón_I	Jugador 6	si	si	No
patrón_K	Jugador 10	si	si	Si
	Jugador 9	si	si	
	Jugador 11	si	si	
RONDA 2 - ITERACIÓN 3				
patrón_B	Jugador 2	no	si	Si
	Jugador 3	no	si	
	Jugador 5	no	si	
patrón_E	Jugador 8	si	no	No
	Jugador 7	si	si	
patrón_I	Jugador 6	si	si	No
patrón_M	Jugador 10	si	si	No

A.4. Otros equipos participantes en la competición *Coach 2006*

En la competición *Coach 2006* participaron seis equipos. A continuación se describen brevemente los equipos más relevantes de la competición.

- **MRL Coach 2006** [65]: Este equipo, ganador de la competición y procedente de la Universidad Qazvin Azad (Irán), modela el equipo oponente utilizando métodos similares a los propuestos en esta tesis. En la fase de *modelado off-line*, se detectan los eventos futbolísticos y se utiliza el test χ^2 para determinar las acciones más relevantes. A continuación, los eventos más relevantes son comparados con los que ocurren en el partido en la fase de *Detección online* con la finalidad de detectar el patrón. Además, este equipo propone un sistema basado en reglas para incitar al equipo *Oponente* a que realice determinadas acciones para detectar mejor determinados patrones.

- **UT Austin Villa Coach 2006** [118]: Este equipo de la Universidad de Texas obtuvo la primera posición en esta competición en los años 2003 [117] y 2005 [119], y la segunda en el año 2004 y 2006. Este equipo considera dos aspectos importantes en el *Modelado offline*:

  1. La formación de un equipo. Para ello se estima, para cada jugador, su posición base y sus vectores de atracción a la pelota. Como la relación entre la posición de la pelota y la de los jugadores es lineal, se utiliza regresión lineal para estimar estos valores.
  2. Estadísticas del equipo. Para la obtención de estas estadísticas es necesario el reconocimiento de los eventos futbolísticos ocurridos durante el partido. A partir de dichos eventos se calculan todo tipo de estadísticas que pueden dar información sobre el comportamiento del equipo.

  En la *detección on-line* se comparan la formación y estadísticas del equipo observado con la información almacenada en los modelos creados.

- **Mehr Coach 2006** [97]: En la primera fase, este equipo detecta una serie de eventos futbolísticos previamente definidos y además obtiene información sobre el posicionamiento de los jugadores. Estas acciones se convierten en reglas *CLang* y son reducidas y generalizadas dividiendo el campo en diferentes regiones. Para detectar los diferentes comportamientos, se comparan las diferentes reglas obtenidas.

A.5. Resultados de la Competición *Coach 2006*

La puntuación que obtiene un agente *coach* varía en función del número de patrones que éste detecta correctamente y la rapidez con que lo realiza. Además, aquellos comportamientos detectados incorrectamente penalizarán en la puntuación obtenida. Considerando esto, para calcular la puntuación del agente *Coach* de un determinado equipo, se utilizaron las siguientes fórmulas:

$$Puntuación = \alpha \times Puntos\ comport.\ correcto - Penaliz.\ comport.\ incorrecto$$

$$Puntos\ comport.\ correcto = 3000 \times N_c + \sum_{i=1}^{N_c} (6000 - tc_i)$$

$$Penalización\ comport.\ incorrecto = 3000 \times N_i + \sum_{j=1}^{N_i} (6000 - ti_j)$$

donde:

$$\alpha = \frac{Número\ total\ de\ comportamientos - Número\ de\ comportamientos\ activados}{Númerodecomportamientosactivados}$$

N_c = Número de comportamientos reportados correctamente
tc_i = Ciclo en el que se ha reportado el comportamiento *i-ésimo* correctamente

N_i = Número de comportamientos reportados correctamente
ti_j = Ciclo en el que se ha reportado el comportamiento *j-ésimo* correctamente

De acuerdo con estas fórmulas, la Tabla A.7[1] muestra la puntuación y los rankings de los equipos en la ronda 1. Este ranking muestra que los resultados de M-COMP dependen mucho del tipo de patrones que se desea reconocer.

Tabla A.7: Resultados competición *Coach 2006*. Primera Ronda

		Iteración 1		Iteración 2		Iteración 3	
Clasif.	Equipo	#	Puntos	#	Puntos	#	Puntos
1	Mehr (Iran)	1	34750.0	1	14300.0	2	34700.0
2	MRL (Iran)	3	11900.0	4	11289.6	1	48789.6
3	**CAOS (España)**	2	25051.2	3	13091.5	5	957.3
4	Austin Villa (USA)	4	5434.2	2	14054.4	4	8312.5
5	Caspian (Iran)	5	-8000.0	5	-8000.0	3	9600.0
6	Pasagard (Iran)	6	-10497.0	6	-10497.0	6	-10497.0

[1]Estos resultados están disponibles en http://ssil.uni-koblenz.de/RC06/

La Tabla A.8 muestra la puntuación de todos los equipos obtenida en la segunda ronda. En este caso, la posición alcanzada utilizando M-COMP fue quinta. Esta posición se debe principalmente a que la mayoría de los patrones a reconocer no estaban relacionados con las acciones realizadas por el poseedor del balón.

		Iteración 1		Iteración 2		Iteración 3	
Clasif.	Equipo	#	Puntos	#	Puntos	#	Puntos
1	Austin Villa (USA)	1	58171.2	1	36188.0	1	97757.2
2	MRL (Iran)	2	17424.0	2	19260.0	2	19386.0
3	Pasagard (Iran)	5	-3097.0	3	9294.0	3	15491.3
4	Caspian (Iran)	4	4800.0	4	-4000.0	4	8000.0
5	**CAOS (España)**	3	11860.0	5	-4480.0	5	0.0
6	Mehr (Iran)	6	-9400.0	6	-29000.0	6	-3000.0

Tabla A.8: Resultados competición *Coach 2006*. Segunda Ronda

Las puntuaciones obtenidas en la ronda final, en la que el equipo *CAOS* no participó, se muestran en la Tabla A.9.

	Iteración 1		Iteración 2		Iteración 3		Iteración 4	
Equipo	#	Puntos	#	Puntos	#	Puntos	#	Puntos
1. MRL (Iran)	1	67583.5	1	40857.6	1	51311.2	1	91578.7
2. Austin Villa (USA)	2	44649.9	4	-4739.4	3	9548.5	2	9543.7
3. Caspian (Iran)	3	11200.0	3	-4000.0	2	15000.0	4	-8000.0
4. Pasagard (Iran)	4	-9282.0	2	2475.2	4	-9282.0	3	5418.0

Tabla A.9: Resultados competición *Coach 2006*. Tercera Ronda

Apéndice B

Introducción a los Modelos Ocultos de Markov

Los modelos ocultos de Markov (HMM del inglés *Hidden Markov Models*) fueron presentados por Baum y Petrie en el año 1966 [23]. El reconocimiento del habla fue una de las primeras tareas en las que se aplicaron estos modelos. Más tarde, los HMM fueron aplicados en el análisis de secuencias biológicas y el reconocimiento de caracteres (OCR) manuscritos.

En este apéndice se presenta una introducción a los modelos ocultos de Markov (HMM) y al *Algoritmo Forward*

B.1. Definición de *Modelos Ocultos de Markov*

Los procesos de *Markov* son modelos matemáticos que describen el comportamiento de un sistema dinámico sometido a un fenómeno de naturaleza aleatoria. Este tipo de procesos son conocidos como procesos estocásticos y tienen gran importancia en el análisis y estudio de sistemas dinámicos.

Una cadena de Markov, también conocida como modelo de Markov observado, es una secuencia de eventos donde la probabilidad de cada uno de ellos depende exclusivamente del evento precedente. Así, una cadena de Markov puede considerarse como un tipo de modelo gráfico probabilístico cuyos estados son observables.

Un HMM es un conjunto finito de estados relacionados mediante una probabilidad de distribución. La transición entre los estados está definida por una serie de probabilidades denominadas *probabilidades de transición*. Pero en este caso, un HMM tiene dos procesos estocásticos embebidos, uno no es observable (estados ocultos), pero puede ser observado mediante el otro proceso (secuencia de observaciones).

La Figura B.1 muestra el esquema de estos dos modelos:

a Las cadenas de Markov, donde *o1*, *o2* y *o3* son estados directamente observables

b Los HMM, donde *q1*, *q2* y *q3* representan los estados ocultos, mientras que *o1*, *o2* y *o3* son los estados observables.

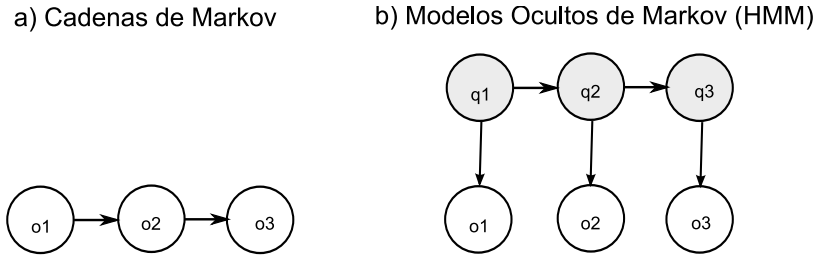


Figura B.1: Esquema de cadenas de Marvok y Modelos Ocultos de Markov (HMM)

B.2. Formalización de los HMM

Para definir un HMM completamente, son necesarios los siguientes elementos:

- Estados ocultos del modelo: Q=$\{q_i\}$, i=1,...N. Cada estado corresponde con un evento no observable.
- Matriz de probabilidades de transición de estados (*A*) en la que se almacenan las probabilidades de pasar del estado *i* al estado *j*: A = $\{a_{ij}$ = P(q_j en el instante *t+1* | q_i en el instante *t*)$\}$, donde P(a|b) es la probabilidad condicional de *a* dado *b*, t=1,...T representa el tiempo y $q_i \in$ Q.
- Símbolos de las observaciones en el alfabeto, *O* = $\{o_k\}$, *k*=1,...,*M*.
- Probabilidad de distribución en cada estado, *B* = $\{b_{ik} = b_i(o_k) = P(o_k|q_i)\}$, donde $o_k \in$ O. De manera informal, B es la probabilidad de que la salida sea o_k dado el estado actual q_i.
- Estado inicial de distribución, Π = $\{p_i$ en el instante t = 1)$\}$.

Para predecir el evento observado en un instante, serán necesarias todas las observaciones pasadas. La siguiente probabilidad condicional formaliza esta búsqueda:

$$P(q_i|\ q_{i-1}\ q_{i-2}\ \ldots\ q_1)$$

Para calcular esta probabilidad, es necesario almacenar valores estadísticos de todas las observaciones, y si la longitud de la secuencia tiene muchos eventos, este cálculo resulta tedioso. Por este motivo, se puede realizar la siguiente simplificación denominada condición de Markov de primer orden:

$$P(q_i|q_{i-1}\ q_{i-2}\ ...\ q_1)\ =\ P(q_i|q_{i-1})$$

En este caso, la condición de Markov se denomina de primer orden porque la probabilidad de un estado depende sólo del estado anterior. Así, sólo se consideran 2 estados y define como N=2.

Si la condición de Markov depende de los dos estados anteriores, N = 3 estados, la probabilidad de un estado se representa como $P(q_i|q_{i-1}\ q_{i-2})$.

De esta forma, se puede expresar la probabilidad conjunta de una secuencia de eventos, aplicando la siguiente suposición de Markov:

$$P(q_i\ q_2\ ...\ q_1) = \prod_{i=1}^{n} P(q_i|q_{i-1})$$

B.3. *Algoritmo Forward*

Este algoritmo recursivo [155] calcula la probabilidad de que una secuencia observada (*O*) pertenezca a un determinado HMM (Θ). Para ello, este algoritmo utiliza la variable $\alpha_t(i)$ que se define como:

$$\alpha_t(i) = P(o_1\ o_2\ ...\ o_t,\ q_t = Q_i \mid \Theta)$$

donde $\alpha_t(i)$ representa la probabilidad de haber observado la secuencia $O = (o_1\ o_2\ ...\ o_t)$ y estar en el estado *i* en el instante *t*, dado el modelo Θ. Así, la probabilidad de la secuencia de observaciones parcial es la suma de $\alpha_t(i)$ sobre todos los estados *N*. Para la realización de este cálculo se debe:

1. Calcular las probabilidades para las secuencias de un único símbolo como el producto del estado inicial *i* y la probabilidad de un símbolo dado o_1 en el estado *i*.

2. Aplicar el algoritmo recursivo del siguiente modo: para calcular $\alpha_{t+1}(j)$, se multiplica cada $\alpha_t(i)$ por la correspondiente probabilidad de transición desde el estado *i* al estado *j*, se suman todos los productos aplicados a todos los estados y por último se multiplica el resultado por la probabilidad del símbolo $o(t + i)$. Realizando este proceso de forma iterativa, es posible calcular $\alpha_t(i)$.

3. La probabilidad requerida se obtiene sumando las probabilidades de todos los caminos que convergen en el estado final N, cuando se ha alcanzado el final de la secuencia observada.

La Figura B.2 expone formalmente el *algoritmo forward*.

Algoritmo Forward

Entradas:
Secuencia observada, *O*.
Modelo (HMM) a considerar, Θ .

Salida:
Probabilidad de que la secuencia *O* pertenezca al modelo Θ

Procedimiento:
// Inicialización:
$\alpha_1(i) = \pi_i b_i(o_1)$, i=1,...,N

// Recursión:
$\alpha_{t+1}(i) = [\sum_{j=1}^{N}(\alpha_t(j)a_{ji})b_i](o(t+1))$,
i=1,...,N;
t=1,...,T-1

// Terminación:
$P(O|\Theta) = \sum_{j=1}^{N}(\alpha_T(j))$

Figura B.2: *Algoritmo Forward* - Calcula la probabilidad de que una secuencia pertenezca a un modelo concreto.

Apéndice C

Cálculo Recursivo del Potencial Utilizando Distancia Coseno

El potencial de un ejemplo en un Espacio de Características (EC) se calcula a partir de las distancias acumuladas entre dicho ejemplo y todos aquellos ejemplos almacenados en el EC. Este valor mide la densidad de datos que lo rodean y puede calcularse utilizando diferentes funciones, pero siempre deben ser monótonas e inversamente proporcionales a la distancia entre todos los ejemplos. A partir de esto, las dos formas más comunes de calcularse se reflejan en las ecuaciones C.1 y C.2. Es decir, considerando o no el cuadrado de la distancia.

$$P_k(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1} distancia(x_k, x_i)}{k-1}} \tag{C.1}$$

$$P_k(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1} distancia^2(x_k, x_i)}{k-1}} \tag{C.2}$$

En estas ecuaciones se observa que el cálculo del potencial de un ejemplo necesita el cálculo de la suma de las distancias de todos ejemplos del EC. Esto implica la necesidad de almacenar todos los ejemplos por lo que es necesaria una cantidad de memoria elevada y la realización de un número elevado de operaciones. Con la finalidad de no necesitar almacenar todos los ejemplos recibidos, en este apéndice se desarrolla y explica cómo se calcula la expresión recursiva del potencial considerando cada una de las dos ecuaciones anteriores.

La expresión del potencial utilizada en esta tesis es la representada por la ecuación C.1. Considerando esta expresión, Angelov y Zhou [13] proponen una ecuación recursiva capaz de calcular el potencial de un ejemplo. El desarrollo de esta ecuación se detalla en el apartado C.1. La importancia de este cálculo reside en que el resultado de dicha expresión es exactamente el mismo que si se almacenaran todos los ejemplos y se aplicara la ecuación C.1. La diferencia entre la ecuación

C.1 y la ecuación recursiva es que ésta última sólo necesita almacenar un conjunto muy reducido de datos.

El potencial también es comúnmente calculado como se muestra en la ecuación C.2. La expresión recursiva que utiliza esta ecuación se propone en esta tesis doctoral como aportación a los SAA. El desarrollo de dicha aportación se detalla y explica en el apartado C.2.

C.1. Expresión Recursiva del Potencial (sin el cuadrado de la Distancia Coseno)

La expresión recursiva que se muestra en este apartado, ha sido obtenida por Angelov y Zhou [13]. Sin embargo, hay pequeñas variaciones en esta demostración frente a la mostrada en [13] debido a la corrección de erratas de notación en el estudio original.

La ecuación del potencial para un ejemplo (vector x_k) se considera tal y como se muestra en la ecuación C.3.

$$P_k(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1} d_{cos}(x_k, x_i)}{k-1}} \tag{C.3}$$

donde *k* indica el ordinal del nuevo ejemplo (vector representado por la letra *x*) en el EC. De esta forma, *k*-1 representa el número de ejemplos insertados en el EC y x_k el ejemplo *k*-ésimo insertado. Cada ejemplo (vector) está formado por una serie de datos representados por su posición en el vector ejemplo. El elemento *i* del vector *k*-ésimo (x_k) se representa por: x_k^i.

Por otra parte, la ecuación de la distancia coseno se muestra en la ecuación C.4.

$$d_{cos}(x_k, x_i) = 1 - \frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n} (x_k^j)^2 \sum_{j=1}^{n} (x_i^j)^2}} \tag{C.4}$$

Considerando el potencial y la distancia coseno, se obtiene:

$$P_k(x_k) = \frac{1}{1 + \frac{1}{k-1} \sum_{i=1}^{k-1} [1 - \frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n} (x_k^j)^2 \sum_{j=1}^{n} (x_i^j)^2}}]} \tag{C.5}$$

A partir de esta ecuación, se obtiene:

$$P_k(x_k) = \frac{1}{1 + [\frac{1}{k-1}[(k-1) - \sum_{i=1}^{k-1}[\frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n}(x_k^j)^2}\sqrt{\sum_{j=1}^{n}(x_i^j)^2}}]]]} \tag{C.6}$$

$$P_k(x_k) = \frac{1}{1 + 1 - [\frac{1}{k-1}\sum_{i=1}^{k-1}[\frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n}(x_k^j)^2}\sqrt{\sum_{j=1}^{n}(x_i^j)^2}}]]} \tag{C.7}$$

$$P_k(x_k) = \frac{1}{2 - \frac{1}{k-1}\frac{1}{\sqrt{\sum_{j=1}^{n}(x_k^j)^2}}\sum_{i=1}^{k-1}\frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n}(x_i^j)^2}}}; k = 2, 3, ...; P_1(x_1) = 1 \tag{C.8}$$

Para mostrar esta ecuación más legible, B_k se denota por la siguiente expresión:

$$B_k = \sum_{i=1}^{k-1} \frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n}(x_i^j)^2}} \tag{C.9}$$

B_k se simplifica en:

$$B_k = \sum_{i=1}^{k-1} \frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n}(x_i^j)^2}} = \sum_{i=1}^{k-1} \frac{1}{\sqrt{\sum_{j=1}^{n}(x_i^j)^2}} \sum_{j=1}^{n} x_k^j \sqrt{(x_i^j)^2} \tag{C.10}$$

Y finalmente:

$$B_k = \sum_{j=1}^{n} x_k^j \sum_{i=1}^{k-1} \sqrt{\frac{(x_i^j)^2}{\sum_{l=1}^{n}(x_i^l)^2}} \tag{C.11}$$

Así, si definimos b_k^i, es decir, si definimos cada uno de los elementos del vector *b*, se obtiene:

$$b_k^j = \sum_{i=1}^{n} \sqrt{\frac{(x_i^j)^2}{\sum_{l=1}^{n}(x_i^l)^2}} \tag{C.12}$$

Y por lo tanto, se puede calcular de forma recursiva el valor de B_k:

$$B_k = \sum_{j=1}^{n} x_k^j b_k^j \; ; \; b_k^j = b_{(k-1)}^j + \sqrt{\frac{(x_k^j)^2}{\sum_{l=1}^{n}(x_k^l)^2}} \; ; \; i = [1, n+1] \tag{C.13}$$

De forma que el primer valor b_1^j es calculado del siguiente modo:

$$b_1^j = \sqrt{\frac{(x_1^j)^2}{\sum_{l=1}^{n}(x_1^l)^2}} \; ; \; i = [1, n+1] \tag{C.14}$$

De esta forma, el potencial de un ejemplo puede ser calculado como se muestra en C.15 y dicho cálculo no requiere almacenar todos los elementos, sino actualizar en cada caso únicamente el vector de datos b_k^j.

$$P_k(x_k) = \frac{1}{2 - \frac{1}{k-1}\frac{1}{\sqrt{\sum_{j=1}^{n}(x_k^j)^2}}\sum_{j=1}^{n} x_k^j b_k^j}; k = 2, 3, ...; P_1(x_1) = 1 \tag{C.15}$$

C.2. Expresión Recursiva del Potencial (con el cuadrado de la Distancia Coseno)

Para obtener la expresión recursiva del potencial utilizando la distancia coseno, se debe combinar la ecuación del potencial para un ejemplo (Ecuación C.16) y la ecuación de la distancia coseno (Ecuación C.4).

$$P_k(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1} d_{cos}^2(x_k, x_i)}{k-1}} \tag{C.16}$$

donde x_k indica el ordinal *k* del nuevo vector (representado por la letra *x*) en el EC. De esta forma, *k*-1 representa el número de vectores insertados en el EC y x_k el vector *k*-ésimo obtenido. Cada vector está formado por una serie de datos que también estarán representados por un número, de esta forma, el elemento *i* del vector (*x*) *k*-ésimo se representa por: x_k^i.

A partir de estas dos ecuaciones (potencial y distancia coseno), se obtiene:

$$P_k(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1}[1 - \frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n}(x_k^j)^2 \sum_{j=1}^{n}(x_i^j)^2}}]^2}{k-1}} \tag{C.17}$$

De forma más sencilla, se representa como:

$$P_k(x_k) = \frac{1}{1 + [\frac{1}{k-1}\sum_{i=1}^{k-1}[1 - \frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n}(x_k^j)^2 \sum_{j=1}^{n}(x_i^j)^2}}]^2]} \tag{C.18}$$

Para explicar la derivación obtenida paso a paso, se considera de forma separada el denominador de la ecuación C.18 que se nombrarán como $denom.P(x_k)$.

$$denom.P_k(x_k) = 1 + [\frac{1}{k-1}\sum_{i=1}^{k-1}[1 - \frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n}(x_k^j)^2 \sum_{j=1}^{n}(x_i^j)^2}}]^2] \tag{C.19}$$

$$denom.P_k(x_k) = 1 + [\frac{1}{k-1}\sum_{i=1}^{k-1}[1 - \frac{f_i}{h\ g_i}]^2] \tag{C.20}$$

donde:

$f_i = \sum_{j=1}^{n} x_k^j x_i^j$

$h = \sqrt{\sum_{j=1}^{n}(x_k^j)^2}$

$g_i = \sqrt{\sum_{j=1}^{n}(x_i^j)^2}$

Se observa que las variables denominadas f_i y g_i dependen del sumatorio que engloba todos los ejemplos necesarios para el cálculo de la distancia coseno y cuyos valores son recorridos utilizando la variable *i*. Sin embargo, la variable *h* sólo representa el valor obtenido de sumar todos los valores del vector obtenido. Considerando esto, se obtiene:

$$denom.P_k(x_k) = 1 + [\frac{1}{k-1}\sum_{i=1}^{k-1}[1 - \frac{2f_i}{h\ g_i} + (\frac{f_i}{h\ g_i})^2]] \tag{C.21}$$

$$denom.P_k(x_k) = 1 + [\frac{1}{k-1}[(k-1) + \sum_{i=1}^{k-1}[-\frac{2f_i}{h\ g_i} + \frac{f_i^2}{h^2\ g_i^2}]]] \tag{C.22}$$

$$denom.P_k(x_k) = 2 + [\frac{1}{k-1}\frac{1}{h}\sum_{i=1}^{k-1}[-\frac{2f_i}{g_i} + \frac{f_i^2}{h\ g_i^2}]] \tag{C.23}$$

$$denom.P_k(x_k) = 2 + [\frac{1}{h(k-1)}[[-2\sum_{i=1}^{k-1}\frac{f_i}{g_i}] + [\frac{1}{h}\sum_{i=1}^{k-1}(\frac{f_i}{g_i})^2]]] \tag{C.24}$$

En este caso, se quiere obtener una expresión recursiva para no almacenar todos los datos obtenidos y por lo tanto, no tener que realizar los dos sumatorios de la ecuación C.24. Por lo tanto, se analizan dichos sumatorios para obtener su valor de forma recursiva. El primero de ellos se denomina B_k y el segundo D_k. Considerando esta notación, se obtiene:

$$denom.P_k(x_k) = 2 + [\frac{1}{h(k-1)}[[-2B_K] + [\frac{1}{h}D_k]]] \tag{C.25}$$

donde:

$B_k = \sum_{i=1}^{k-1} \frac{f_i}{g_i}$

$D_k = \sum_{i=1}^{k-1} (\frac{f_i}{g_i})^2$

A continuación se analiza cada sumatorio por separado:

1. Si se analiza el primero de los sumatorios (B_k), y se considera el cambio de nombre realizado en la ecuación C.20, se obtiene:

$$B_k = \sum_{i=1}^{k-1} \frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n} (x_i^j)^2}} \tag{C.26}$$

De esta forma, esta ecuación puede ser simplificada del siguiente modo:

$$B_k = \sum_{i=1}^{k-1} \frac{1}{\sqrt{\sum_{j=1}^{n} (x_i^j)^2}} \sum_{i=1}^{k-1} \sum_{j=1}^{n} x_k^j \sqrt{(x_i^j)^2} \tag{C.27}$$

Y finalmente:

$$B_k = \sum_{j=1}^{n} x_k^j \sum_{i=1}^{k-1} \sqrt{\frac{(x_i^j)^2}{\sum_{l=1}^{n} (x_i^l)^2}} \tag{C.28}$$

Así, si definimos b_k^i, es decir, si definimos cada uno de los elementos del vector *b*, se obtiene:

$$b_k^j = \sum_{i=1}^{n} \sqrt{\frac{(x_i^j)^2}{\sum_{l=1}^{n} (x_i^l)^2}} \tag{C.29}$$

Y por lo tanto, se puede calcular de forma recursiva el valor de B_k:

$$B_k = \sum_{j=1}^{n} x_k^j b_k^j \; ; \; b_k^j = b_{(k-1)}^j + \sqrt{\frac{(x_k^j)^2}{\sum_{l=1}^{n}(x_k^l)^2}} \tag{C.30}$$

El primer valor b_1^j es calculado del siguiente modo:

$$b_1^j = \sqrt{\frac{(x_1^j)^2}{\sum_{l=1}^{n}(x_1^l)^2}} \tag{C.31}$$

2. Si se analiza el segundo de los sumatorios (D_k), se obtiene:

   $D_k = \sum_{i=1}^{k-1}(\frac{f_i}{g_i})^2$

   Si de nuevo se considera el renombramiento realizado en la ecuación C.20, se obtiene:

$$D_k = \sum_{i=1}^{k-1}(\frac{\sum_{j=1}^{n} x_k^j x_i^j}{\sqrt{\sum_{j=1}^{n}(x_i^j)^2}})^2 \tag{C.32}$$

   La ecuación recursiva que considere los sumatorios es la siguiente:

$$D_k = \sum_{i=1}^{k-1} \frac{(\sum_{j=1}^{n} x_k^j x_i^j)^2}{(\sqrt{\sum_{j=1}^{n}(x_i^j)^2})^2} = \sum_{i=1}^{k-1} \frac{1}{\sum_{j=1}^{n}(x_i^j)^2}(\sum_{j=1}^{n} x_k^j x_i^j)^2 \tag{C.33}$$

$$D_k = \sum_{i=1}^{k-1} \frac{1}{\sum_{j=1}^{n}(x_i^j)^2}(\sum_{j=1}^{n} x_k^j x_i^j)(\sum_{p=1}^{n} x_k^p x_i^p) = \sum_{j=1}^{n} x_k^j \sum_{p=1}^{n} x_k^p \sum_{i=1}^{k-1} x_k^{ij} x_k^{ip} \frac{1}{\sum_{l=1}^{n}(x_i^l)^2} \tag{C.34}$$

   Al igual que se ha realizado anteriormente, para obtener la expresión recursiva, se considera:

$$d_k^{jp} = \sum_{i=1}^{k-1} x_k^{ij} x_k^{ip} \frac{1}{\sum_{l=1}^{n}(x_i^l)^2} \tag{C.35}$$

Y por lo tanto:

$$D_k = \sum_{j=1}^{n} x_k^j \sum_{p=1}^{n} x_k^p d_k^{jp} \tag{C.36}$$

donde

$$d_k^{jp} = d_{(k-1)}^{jp} + \frac{x_k^j x_k^p}{\sum_{l=1}^{n}(x_k^l)^2}; d_1^{j1} = \frac{x_1^j x_1^1}{\sum_{l=1}^{n}(x_1^l)^2} \; ; \; j = [1, n+1] \tag{C.37}$$

Por último, si se agrupa todo lo obtenido hasta ahora, se obtiene:

$$P_k(x_k) = \frac{1}{2 + [\frac{1}{h(k-1)}[[-2B_K] + [\frac{1}{h}D_k]]]}; k = 2, 3, ...; P_1(x_1) = 1 \tag{C.38}$$

donde B_k se obtiene de forma recursiva como se muestra en las ecuaciones C.30 y C.31, y D_k tal y como se describe en la ecuación C.37.

Se puede observar en las ecuaciones C.36 y C.37 que para obtener de forma recursiva el valor D_k, a diferencia de la ecuación recursiva de B_k, se debe almacenar el producto obtenido de multiplicar cada uno de los valores del vector con el resto de valores. De esta forma, estos resultados deberán almacenarse en una matriz; a diferencia de la obtención de B_k que podía obtenerse almacenando únicamente un vector de datos. Los valores de esta matriz se indican con dos superíndices, por ejemplo el valor d_k^{ij} indica el valor de la fila *i* y columna *j* del vector d_k. Esto supone que si el vector que se desea representar en el EC tiene una cantidad elevada de elementos, la cantidad de valores que se deben almacenar para poder calcular su potencial de forma recursiva es muy elevada (el cuadrado de dicha cantidad).